Escuela Técnica Superior De Ingenieros Industriales y de Telecomunicación.

Modelado Computacional de un Banco de Ensayos para un Bogie de Tren a Escala del laboratorio IMAC.

Máster en Ingeniería Mecánica Aplicada y Computacional.

Trabajo Fin de Estudios.

José David Fuentes Lárez

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Pámplona, 12 de Febrero de 2016

Agradecimientos

En primer lugar, agradecer a todas las personas que han estado involucradas de alguna manera en este proyecto y que no solo me han dado su apoyo sino el ánimo de continuar, no ha sido un camino fácil, sin embargo solo de estos es donde el aprendizaje queda, a mis amigos en especial a Javier Lopez quien me acompaño en cada paso dado en este Máster y que nunca dejo de darme su apoyo aun en los momentos mas difíciles, a mis compañeros del curso quienes me han mostrado un compañerismo que va mas allá de todo, a mi amiga Esti que esta muy molesta conmigo (estoy seguro que se le va a pasar), y a todos los profesores involucrados en especial Javier Ros y Aitor Plaza, quienes se dieron a la tarea de transmitir sus conocimientos de manera abnegada en todo momento y por lo cual tienen mi aprecio, y para terminar, a las personas mas importantes, con las que con mucha suerte algún ser humano pueda contar, y creo que tuve la suerte de tener a los mas especiales, a mis hermanos y mis padres quienes no paran de darme su apoyo todos los días y que en la distancia se las arreglaron para estar conmigo y apoyarme, no me queda mas sino decir: Gracias!

Abstract

In this work we show a Multibody model for the dinamic simulation of the testing bench of a scaled railway of the laboratory IMAC. Taking an special account to the characterization of the forces and moments that is being produced in the contact point between the wheelset and the rail. so the representation of the bogie could be as accurate as posible.

The contact model used in the present work is the one presented by the Kalker linear theory, wich relates the creeps produced in the contact point between the wheels with the tangencial forces and moments produced in this point trough certains values and ecuations. To find the numeric solution in face of the diferencial ecuations obtained of the dinamics , has been precised the use of the Euler implicit Linear kalker model which is the aplication of Euler explicit for all the model except for the introduction of the Kalker forces that solves in the model in the implicit way.

Once we finish the model, its importat to know that its possible keep including a more number or functions or efects, that will become more accurate , realistic and capable to adapt to a mayor number of situations.

keywords :KALKER,Contact,Wheel-Rail,Hertz ,Model, Multibody,Simulation.

Resumen

En este Trabajo se presenta un modelo Multibody para la simulación de la dinámica del banco de ensayos de bogie de tren a escala del laboratorio IMAC. Prestando especial atención a la caracterización de las fuerzas y momentos que se producen en el punto de contacto de la rueda con el raíl, de forma que el comportamiento del Bogie sea lo más preciso posible.

El modelo de contacto utilizado es el presentado por la teoría lineal de Kalker la cual relaciona los pseudo deslizamientos producidos en el punto de contacto entre las ruedas, con las fuerzas tangenciales y momento producidos en este punto a través de ciertos parámetros y ecuaciones. Para hallar la solución numérica ante las ecuaciones diferenciales obtenidas de la dinámica, ha sido preciso utilizar el método de integración de Euler explicito-implícito el cual es explicito para todo excepto para las fuerzas de Kalker que se trabajan de manera implícita en el modelo.

Una vez terminado el planteamiento del modelo, es importante tener en cuenta que es posible seguir incluyendo un mayor número de efectos que lo harían ser más preciso, más realista y capaz de adaptarse a un mayor número de situaciones.

Palabras claves :KALKER,Contacto,Rueda-Rail,Hertz ,Modelo, Multibody,Simulador.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Introducción

Hoy día la simulación numérica de sistemas dinámicos es un tema de interés especial para el mundo científico puesto que es usada en la industria, para predecir de alguna manera los efectos de esfuerzos y cargas a los que un elemento compuesto por diferentes cuerpos puede estar sometido.

En el laboratorio de Ingeniería Mecánica Aplicada y Computacional se encuentra un banco de ensayos de un bogie de tren a escala desarrollado en un trabajo de fin de grado con el fin de poder experimentar y obtener parámetros dinámicos del sistema.

Figura 1.1: Banco de ensayos

Un paso para esto es desarrollar un modelo dinámico que ayude a experimentar e identificar los parámetros dinámicos del conjunto.

La caracterización fiable de las fuerzas de contacto de fricción que aparecen entre el raíl y la rueda fue una de las prioridades del proyecto. Para la simulación dinámica el modelo de contacto es de gran importancia puesto que no solo tiene que describir de manera muy exacta el comportamiento del sistema sino que también tiene que ser de alguna manera discreto para que su cálculo sea óptimo en el tiempo.

Un modelo con el cual se recogen estas características es el modelo de contacto lineal de Kalker con el cual se cuenta con una serie de parámetros obtenidos a partir de la huella y de la fuerza normal ejercida entre el raíl y la rueda.

La elaboración de un simulador que fuese capaz de representar el comportamiento del banco de ensayos correctamente en situaciones diferentes tiene cierto grado complejidad. La precisión y capacidad de adaptación tienen como inconveniente un gran coste computacional - el método de integración es el método de Euler explicito-implícito con un paso de integración de 0.001 s. Esto se traduce que el tiempo de análisis es mayor que el tiempo físico del movimiento, por lo que no es posible realizar simulaciones a tiempo real.

1.2. Objeto

El objeto de este proyecto es modelar, el banco de ensayos de un bogie de tren a escala reducida mediante las técnica de sistemas Multi-cuerpo y el uso de los programas lib_3D_mec y Matlab, con los cuales podamos simular o reproducir diferentes condiciones de trabajo. Los procesos que se pretenden simular son:

1. Simulación de condiciones de trabajo estándar.
2. Obtención de las fuerzas de contacto de Kalker en cualquier instante.
3. Simulación y obtención de parámetros bajo diferentes condiciones de trabajo en cualquier instante de tiempo.

Los aspectos más importantes en el desarrollo del proyecto son:

1. Implementación del modelado del sistema en la librería lib_3D_mec.
2. Resolución del punto de contacto entre el raíl y la ruedas.
3. Implementación del modelo lineal de contacto de Kalker.
4. Simulación de la evolución en el tiempo modelo.(simulación dinámica)

Las características del modelo son:

1. Posibilidad de reproducir diversas condiciones de trabajo.
2. Posibilidad de obtener la posición del sistema en cualquier instante.

3. Posibilidad de obtener las fuerzas que se producen debido a la fricción en cualquier instante de tiempo.

En este proyecto se ha realizado toda la parte correspondiente al modelado de los elementos en lib_3D_mec y su simulación mediante sistemas multicuerpos. Además de la incorporación del modelo de contacto de Kalker.

1.3. Sistemas Multi-cuerpo

Las técnicas de la Dinámica de Sistemas Multicuerpo (DSM) permiten la simulación de cualquier sistema o subsistema mecánico, y con ello su análisis, diseño y mejora. Resulta claro por tanto el interés industrial, económico y científico de la DSM.

Por lo cual, para comprender los procedimientos y las técnicas aplicadas, procedemos a definir los términos a continuación

1.3.1. Cinemática

Habitualmente las ecuaciones de un sistema mecánico pueden plantearse en términos analíticos gracias a la definición de la posición del sistema en términos de un conjunto de variables constantes en el tiempo denominadas parámetros

$$\mathbf{p} = \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix} \tag{1.1}$$

Y de variables que varían con el tiempo denominadas coordenadas generalizadas:

$$\mathbf{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix} \tag{1.2}$$

Y cuyo objeto es describir en cada instante de tiempo, la posición del sistema mecánico. Las coordenadas generalizadas, $\mathbf{q}$, se relacionan mediante un conjunto de ecuaciones geométricas o ecuaciones para las coordenadas generalizadas. Dichas relaciones, en general no lineales quedan recogidas en un vector ϕ de tal forma que:

$$\mathbf{\Phi}(\mathbf{q}, t) = \begin{bmatrix} \Phi_1(\mathbf{q}, t) \\ \vdots \\ \Phi_n(\mathbf{q}, t) \end{bmatrix} \tag{1.3}$$

1.3.2. Ecuaciones Cinemáticas

Formalmente, las ecuaciones geométricas no son más que relaciones arbitrarias entre las coordenadas $\mathbf{q}$, la variable tiempo, t, y un conjunto de parámetros relacionados con la geometría del sistema mecánico. Desde un punto de vista operativo, las ecuaciones geométricas se plantean como condiciones que deben cumplir los vectores de posición de puntos y vectores unitarios de las bases introducidas para posicionar los elementos del sistema.

1.3.3. Ecuaciones Cinemáticas para las Velocidades Generalizadas

Ecuaciones holónomas: la derivada de las ecuaciones geométricas denotaran las relaciones que deben existir entre las velocidades generalizadas, relaciones que son lineales en las incógnitas $\dot{\mathbf{q}}$.

Además pueden existir relaciones adicionales entre las velocidades, que también son lineales en estas, a esto denominamos ecuaciones no-holónomas y se diferencian de las anteriores ya que estas no tienen un origen geométrico provienen de relaciones que deben satisfacer las velocidades de los puntos del sistema las cuales pueden expresarse de la siguiente forma:

$$\mathbf{A}(\mathbf{q},\mathbf{t})\dot{\mathbf{q}} + \mathbf{b}(\mathbf{q},t) = 0 \tag{1.4}$$

Las cuales darán lugar a un sistema de ecuaciones para las velocidades generalizadas:

$$\mathbf{\Phi_v}(\mathbf{q},\mathbf{t})\dot{\mathbf{q}} + \mathbf{b_v}(\mathbf{q},t) = 0 \tag{1.5}$$

Donde:

$$\mathbf{\Phi_v} = \begin{bmatrix} \mathbf{\Phi_q} \\ A \end{bmatrix} y \begin{bmatrix} \mathbf{\Phi_t} \\ b \end{bmatrix} \tag{1.6}$$

Φ_v como el jacobiano del problema de velocidad. b_v el término independiente del problema de velocidad.

A partir de esto definimos el problema de montaje como aquel que busca las coordenadas generalizadas que son compatibles con las ecuaciones geométricas

1.3.4. Ecuaciones Cinemática para las Aceleraciones Generalizadas:

La derivada total de las ecuaciones anteriores respecto al tiempo nos produce las ecuaciones que relacionan las aceleraciones generalizadas, según:

$$\mathbf{\Phi_v}(\mathbf{q},\mathbf{t})\dot{\mathbf{q}} + \mathbf{\Phi_v}(\mathbf{q},\dot{\mathbf{q}},\mathbf{t})\dot{\mathbf{q}} + \mathbf{b_v}(\mathbf{q},\dot{\mathbf{q}},\mathbf{t}) = \mathbf{0} \tag{1.7}$$

donde:

$\dot{\Phi}_v$ es la derivada del jacobiano del problema de velocidad respecto al tiempo

$$\dot{\mathbf{\Phi}}_{\mathbf{v}} = \sum_{\mathbf{j-1}}^{\mathbf{p}} \frac{\partial}{\partial \mathbf{q_i}} \mathbf{\Phi_v} \dot{\mathbf{q}}_{\mathbf{j}} + \frac{\partial}{\partial \mathbf{t}} \mathbf{\Phi_v} \tag{1.8}$$

$\dot{b_v}$ representa la derivada del término independiente del problema de la velocidad que formalmente se define como:

$$\dot{\mathbf{b}}_{\mathbf{v}} = \begin{bmatrix} \frac{\partial \mathbf{\Phi_t}}{\partial \mathbf{q}} \dot{\mathbf{q}} + \frac{\partial \mathbf{\Phi_t}}{\partial_{\mathbf{t}}} \\ \frac{\partial \mathbf{b}}{\partial \mathbf{q}} \dot{\mathbf{q}} + \frac{\partial \mathbf{b}}{\partial_{\mathbf{t}}} \end{bmatrix} \tag{1.9}$$

1.3.5. Tipos de Coordenadas

La elección de las coordenadas es una delicada tarea que no tiene una respuesta única. El tipo de coordenadas, junto con el formalismo empleado, determinarán la estructura final de las ecuaciones.

Problema de montaje para velocidades y aceleraciones generalizadas: De la misma forma que los valores iniciales dados a las coordenadas generalizadas no tienen por qué satisfacer el conjunto de ecuaciones geométricas, los valores inicialmente asignados a las velocidades generalizadas tampoco tienen por qué satisfacer las ecuaciones cinemáticas para las velocidades generalizadas. Lo mismo puede decirse de los valores inicialmente asignados a las aceleraciones generalizadas.

Forma más usual de resolver este sistema de ecuaciones no lineales es utilizar el método de Newton-Raphson, El método de Newton-Raphson sustituye el sistema no lineal de ecuaciones $\phi(\mathbf{q}; \mathbf{t}) = 0$ por una aproximación en serie de Taylor de primer orden respecto de las incógnitas.

$$\mathbf{\Phi}(\mathbf{q}, \mathbf{t}) \approx \mathbf{\Phi}(\mathbf{q^0}, \mathbf{t}) + \mathbf{\Phi_q}(\mathbf{q^0}, \mathbf{t})(\mathbf{q} - \mathbf{q^0}) = \mathbf{0} \tag{1.10}$$

El sistema presentara una solución aproximada puesto a fin de cuentas es una aproximación en serie de Taylor. Sin embargo este sistema no tiene solución única por lo tanto se tienen diferentes maneras de no tener una solución única pero sí la más adecuada, la más común se tiene como la solución de norma mínima .

La solución de norma mínima de un sistema lineal de ecuaciones indeterminado de la forma general.

$$\mathbf{Ax} = \mathbf{b} \tag{1.11}$$

Es aquella que busca dentro de las infinitas soluciones aquella que tiene la mínima norma. Así la solución de norma mínima es aquel vector $\mathbf{x}$ de la afinidad que es normal al sub-espacio trasladado.

Otra solución son las coordenadas dependientes en términos de independientes, en cada iteración de Newton Raphson elegir un conjunto de coordenadas como independientes, dejar su valor sin modificar y determinar el valor del resto en términos de estas. Este algoritmo funciona bien si el método de elegir las coordenadas independientes es adecuado.

El método de elección de las coordenadas independientes están ligados al método de solución. Por lo que para llegar a esto es posible pensar en la eliminación gaussiana. La cual consiste en hacer 0 los elementos situados debajo de la diagonal de la matriz del sistema. Lo cual triangulará la matriz. La estrategia numérica consiste en el movimiento de filas y columnas que permite elegir el elemento de la diagonal que se utilizara para hacer ceros por debajo de forma que se produzca el mínimo error en la solución.

Con lo cual esta estrategia de pivoteo de filas y columnas propone una forma óptima de variables dependientes e independientes. A esta separación de coordenadas se le denomina partición de coordenadas.

1.3.6. Problemas de posición, velocidad y aceleración iniciales

Para obtener el valor de las coordenadas generalizadas, de forma que el sistema se encuentre montado y en la posición deseada, se aplica el algoritmo de Newton Raphson . Análogamente las ecuaciones cinemáticas para las velocidades no forman un conjunto completo de ecuaciones. Así si se desea inicializar las velocidades de forma que su valor sea compatible con ellas es necesario introducir un conjunto adicional de ecuaciones para que el sistema resultante sea compatible determinado.

El sistema lineal de ecuaciones resultantes puede resolverse utilizando el método de eliminación gaussiana. Evidentemente esto también puede aplicarse para las aceleraciones generalizadas

1.3.7. Simulación cinemática

Con frecuencia suele ser interesante simular el movimiento del mecanismo en un contexto puramente cinemático.

La simulación cinemática para sistemas no-holónomos requiere la integración numérica. Para la solución de estos sistemas consideramos el método de Euler en donde en cada instante de tiempo la solución de q se puede sustituir por su aproximación en serie de Taylor de primer orden.

$$\mathbf{q}(\mathbf{t}+\mathbf{\Delta t}) \approx \mathbf{q}(\mathbf{t}) + \dot{\mathbf{q}}(\mathbf{t})\mathbf{\Delta t} \tag{1.12}$$

$$\dot{\mathbf{q}}(\mathbf{t}+\mathbf{\Delta t}) \approx \dot{\mathbf{q}}(\mathbf{t}) + \ddot{\mathbf{q}}(\mathbf{t})\mathbf{\Delta t} \tag{1.13}$$

Al disponer del valor de $\dot{\mathbf{q}}$ podemos realizar la integración de manera más óptima sustituyendo de manera ventajosa:

$$\mathbf{q} = \mathbf{q} + (\dot{\mathbf{q}}\mathbf{\Delta t} + \frac{\mathbf{1}}{\mathbf{2}}\ddot{\mathbf{q}})\mathbf{\Delta t} \tag{1.14}$$

1.3.8. Dinámica

De forma clásica, el planteamiento de las ecuaciones dinámicas de cualquier sistema, requiere la caracterización de las acciones que recaen sobre los diferentes elementos del mismo.

Con el objeto de introducir la notación utilizada en las diferentes secciones del capítulo, se propone la siguiente clasificación:

Acciones de inercia: son aquellas acciones derivadas de las fuerzas y momentos de inercia de D'Alembert

Acciones de enlace: son aquellas acciones encargadas de mantener las ligaduras cinemáticas entre los diferentes elementos del sistema. Y su efecto queda reflejado en las ecuaciones dinámicas a través de las denominadas incógnitas de enlace,

En el caso particular de sistemas de sólidos, éstas se subdividen en:

- Acciones correspondientes a enlaces compatibles con la acotación de partida.
- Acciones correspondientes a enlaces no respetados por la acotación de partida.

Esta clasificación tiene su importancia cuando se implementan formulaciones de tipo lagrangiano, ya que en estos casos, la contribución de las primeras se anula en virtud de la tercera ley de Newton.

Acciones constitutivas: son aquellas acciones que vienen caracterizadas por una ley constitutiva.

Otras acciones: son aquellas que no pueden englobarse en ninguno de los apartados anteriores y se denotan de la misma forma que las constitutivas con el objeto de diferenciarlas de las de enlace.

En el caso del solido rígido las ecuaciones vectoriales son el teorema del momento lineal para un sistema de partículas o ecuaciones de Newton y el teorema del momento angular o cinético o ecuaciones de Euler. Esto hace que las citadas ecuaciones se conozcan comúnmente como ecuaciones de Newton-Euler.

$$-\frac{\mathbf{d\tilde{D}}(Sol_k)}{dt}\Big|_{\mathbf{R.I}} + \sum_{\mathbf{L_k}}^{\mathbf{j-1}} \tilde{\mathbf{F}}_{\mathbf{j}}^{\mathbf{sol_k}} = \mathbf{0} \tag{1.15}$$

$$-\frac{\mathbf{d\tilde{H}_{Gk}}(Sol_k)}{dt}\Big|_{\mathbf{R.I}} + \sum_{\mathbf{L_k}}^{\mathbf{j-1}} \mathbf{M}\tilde{\mathbf{j}}_{\mathbf{GK}}^{\mathbf{sol_k}} = \mathbf{0} \tag{1.16}$$

Donde K hace referencia al número de sólidos L_k es el número de acciones exteriores al solidos sol_k,G_k es el centro de gravedad del sólido sol_k ,R.I. es una referencia inercial, $\mathbf{d\tilde{D}(Sol_k)} = m_{sol_k}\mathbf{\tilde{V}R.I(G_k)}$ es el momento lineal del sólido, $\mathbf{H_{Gk}(\tilde{Sol}_k)} = \mathbf{\bar{\bar{I}}_{G_k}(sol_k)\tilde{\Omega}}R.I(Sol_k)$ en el momento angular del sólido.

El tensor de inercia es un concepto análogo al de momento de inercia utilizado en problemas en dos dimensiones, pero extendido al caso de tres dimensiones.

De hecho siempre existe una base orto normal solidaria al sólido, en la que el momento angular del solido es igual al momento.

$$\left\{\vec{\mathbf{H}_{\mathbf{G_k}}}(Sol_k)\right\} = \begin{Bmatrix} I_1\omega_1 \\ I_2\omega_2 \\ I_3\omega_3 \end{Bmatrix} = \begin{bmatrix} I_1 & 0 & 0 \\ 0 & I_2 & 0 \\ 0 & 0 & I_3 \end{bmatrix} \begin{Bmatrix} \omega 1 \\ \omega 2 \\ \omega 3 \end{Bmatrix} \tag{1.17}$$

Frecuentemente conviene denominar a las parejas Fuerza-Momento como torsor (wrench en inglés) del sistema de fuerzas correspondiente.

$$\begin{bmatrix} \vec{\mathbf{F}}_{\mathbf{j}}{}^{Sol_k} \\ \vec{\mathbf{M}}_{\mathbf{j}}{}^{Sol_k}_{G_k} \end{bmatrix} \tag{1.18}$$

En este contexto, a la pareja Velocidad-Velocidad Angular se denomina rotor (twist en inglés), y puede considerarse un vector dual del torsor, puesto que multiplicado por el escalarmente nos daría la potencia del torsor):

$$\begin{bmatrix} \vec{\mathbf{F}}_{\mathbf{j}}{}^{Sol_k} \\ \vec{\mathbf{M}}_{\mathbf{j}}{}^{Sol_k}_{G_k} \end{bmatrix}^T \begin{bmatrix} \vec{\mathbf{V}}_{(\mathbf{O_k})} \\ \vec{\mathbf{\Omega}}_{(Sol_k)} \end{bmatrix} = \dot{\mathbf{W}}(Action_j, Sol_k) \tag{1.19}$$

El torsor:

$$\begin{bmatrix} \vec{\mathbf{F}}^{Sol_k} \\ \vec{\mathbf{M}}^{Sol_k} \end{bmatrix} = \begin{bmatrix} -\frac{d\mathbf{D}(\vec{Sol}_k)}{dt}\Big|_{R.I} \\ -\frac{d\mathbf{H}_{\mathbf{G_k}}(\vec{Sol}_k)}{dt}\Big|_{R.I} \end{bmatrix} \tag{1.20}$$

Se denomina torsor de acciones de inercia o de acciones de inercia de D'Alembert. Utilizando la notación de torsores empleada anteriormente las ecuaciones para los diferentes solidos pueden escribirse

$$\begin{bmatrix} -\frac{d\mathbf{D}(\vec{Sol}_k)}{dt}\Big|_{R.I} \\ -\frac{d\mathbf{H}_{\mathbf{G_k}}(\vec{Sol}_k)}{dt}\Big|_{R.I} \end{bmatrix} + \sum_{j=1}^{L_k} \begin{bmatrix} \vec{\mathbf{F}}_{\mathbf{j}}{}^{Sol_k} \\ \vec{\mathbf{M}}_{\mathbf{j}}{}^{Sol_k}_{G_k} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, K = 1...., M \tag{1.21}$$

Esta notación con torsores es particularmente conveniente ya que en dinámica de solido rígido: la fuerza y el momento resultantes asociados a una distribución de fuerzas (acción) actuante sobre un sólido rígido son conceptualmente inseparables. No obstante es importante constatar que las citadas aceleraciones no son las únicas incógnitas del problema dinámico. También aparecen como incógnitas en el sistema anterior las incógnitas de enlace. las cuales se definen como aquel conjunto de incógnitas que se utiliza para caracterizar las fuerzas y momentos de enlace actuantes sobre los sólidos del sistema.

1.3.9. Estructura del sistema de Ecuaciones Dinámico

Las ecuaciones dinámicas son lineales en las aceleraciones generalizadas $\ddot{\boldsymbol{q}}$ por ser las derivadas del momento lineal y del momento angular. Por otra parte

las acciones de enlace son lineales en las incógnitas de enlace. Resto de acciones no depende de estas incógnitas así las ecuaciones dinámicas toman la forma general:

$$\begin{bmatrix} \boldsymbol{M_{v\dot{q}}} & \boldsymbol{V_q^{\perp}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\ddot{q}} \\ \boldsymbol{\varepsilon} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\delta} \end{bmatrix} \tag{1.22}$$

donde $V_q^{\perp}$ es definida como la matriz que cumple :

$$\boldsymbol{V_q V_q^{\perp}} = 0 \tag{1.23}$$

Para el caso más normal en el que existan relaciones cinemáticas es necesario completar el sistema con las ecuaciones cinemáticas para las aceleraciones teniendo:

$$\begin{bmatrix} \boldsymbol{M_{V\dot{q}}} & \boldsymbol{V_{\dot{q}}^{\perp}} \\ \boldsymbol{\Phi_{\dot{q}}} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\ddot{q}} \\ \boldsymbol{\varepsilon} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\delta} \\ \boldsymbol{\gamma} \end{bmatrix} \tag{1.24}$$

1.3.10. Estructura del Sistema de Ecuaciones Dinámico

El problema dinámico directo resuelve el sistema de ecuaciones determinando las aceleraciones generalizadas y las incógnitas de enlace. Conocidos los valores de las coordenadas y velocidades generalizadas en un instante de tiempo t. De no existir redundancia en las ecuaciones de enlace el sistema puede resolverse de la siguiente manera:

$$\begin{bmatrix} \boldsymbol{M_{V\dot{q}}} & \boldsymbol{\Phi_q^T} \\ \boldsymbol{\Phi_{\dot{q}}} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\ddot{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\delta} \\ \boldsymbol{\gamma} \end{bmatrix} \tag{1.25}$$

Donde $\boldsymbol{\delta}$ es el conjunto de fuerzas actuantes sobre el sistema sin tomar en cuenta las acciones de enlace, $\boldsymbol{\lambda}$ es un conjunto de incógnitas auxiliares denominadas multiplicadores de Lagrange que pueden obtenerse del sistema una vez conocidas las aceleraciones del sistema anterior.

La solución más sencilla a estos problemas suele ser buscar la solución de norma mínima, para lo cual basta usar la pseudo inversa en vez de la inversa. El resto de incógnitas quedan completamente determinadas. Otra posibilidad es la utilización de una eliminación gaussiana con movimiento de columnas de forma que las incógnitas que no tienen asociado un elemento en la diagonal se toman como independientes. Este algoritmo suele utilizarse utilizando la descomposición LU con pivote.

Capítulo 2

El Modelo

2.1. Descripción del Modelo

El proyecto surge de un trabajo de fin de estudios donde la idea principal era diseñar y construir un prototipo de bogie de tren a escala que permita medir diferentes parámetros relacionados con el movimiento del mismo a lo largo de la vía. Los parámetros reales del tren fueron escalados y llevados al modelo a escala para así poder obtener unos valores que puedan ser llevados y analizados en la realidad.

Los procesos experimentales que se pretenden llevar a cabo con el banco de ensayos son los siguientes:

- Estimación de los parámetros inerciales.
- Estimación del estado, es decir, coordenadas y velocidades generalizadas que definan el movimiento del sistema.
- Estimación de las fuerzas de contacto entre la rueda y la vía.

2.2. Características generales del banco de pruebas

1. Bogie: Se define el bogie como un bastidor que puede girar respecto al chasis del vagón, al que se fija el tren de rodadura. Está formado dos o tres pares de ruedas, montadas sobre sus respectivos ejes, los cuales son paralelos y solidarios entre sí, cuya función es circular sobre los raíles de la vía.

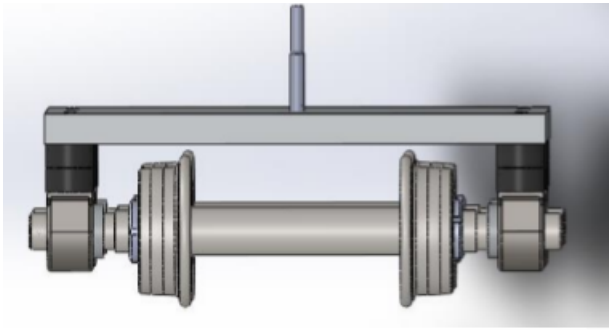

Figura 2.1: Bogie

2. Eje Calado o Whelseet: Conjunto formado por un eje y sus dos ruedas. El eje les proporciona a las ruedas la misma velocidad angular y una distancia constante entre ellas. Este eje establece en el sistema:

   - La distancia entre el vehículo y la vía.
   - Los medios de transmisión de las fuerzas de tracción y de frenado a los raíles para acelerar y decelerar el vehículo.
   - El movimiento de lazo: oscilación que describe el eje al desplazarse de su posición centrada mientras circula el tren de forma libre por la vía.

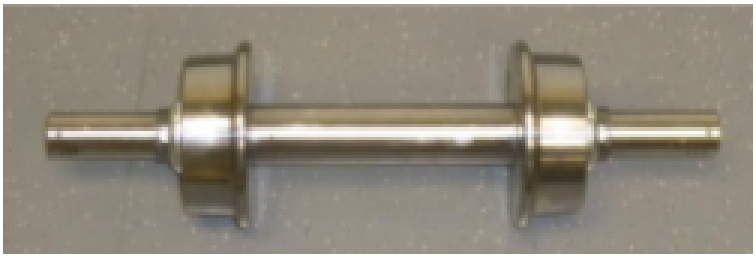

Figura 2.2: Whelseet

3. Bancada:

   El bastidor del bogie tiene forma de H y está formado por planchas de acero soldadas. Tiene la función de formar la base de la estructura del bogie para montar sobre él el resto de componentes. Los materiales más utilizados para su construcción son:

   - SS400 (acero laminado para estructuras generales).
   - SM400B (acero laminado para estructuras soldadas).

Figura 2.3: Bancada

4. Ruedas: Elemento de transmisión que permite:

   - Soportar el peso del vehículo.
   - Guiar al vehículo por la vía.
   - Frenar y acelerar al vehículo.

   El desgaste que se produce en las ruedas que circulan por el carril exterior en la curva, se debe al deslizamiento que se produce al recorrer mayor distancia durante el trazado de curvas. Una solución consiste en fabricar un perfil cónico para la superficie de rodadura de la rueda cuyo ángulo de inclinación sea variable respecto al eje calado.

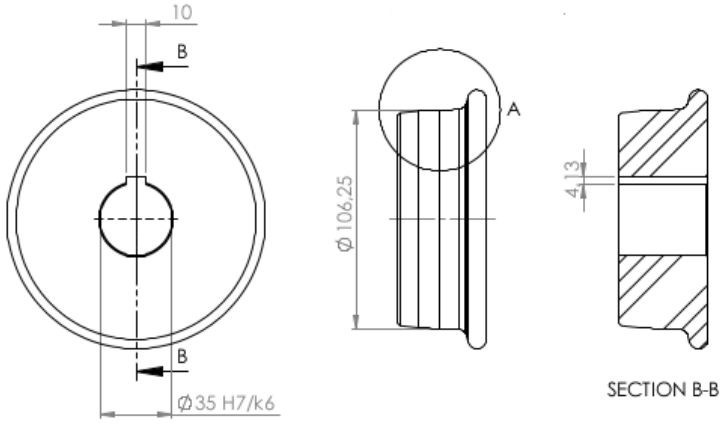

Figura 2.4: Ruedas

5. Vía:

   Se entiende por vía al camino por el que circulan los trenes. Por este motivo, su función consiste en el guiado de los vehículos ferroviarios de la forma más segura y económica posible. Cualquier fallo en la nivelación y alineación de la vía, produce vibraciones en el tren, repercutiendo en el confort de los pasajeros; e incluso se pueden generar daños en la estructura.

6. Raíl:

   Elemento sobre el que circula el vehículo. Sus funciones son las siguientes:

   - Resistir y transmitir las tensiones que recibe del tren.
   - Guiar a las ruedas en su movimiento.
   - Conducir la corriente eléctrica para la señalización en las líneas electrificadas.

   Los elementos que lo componen son:

   - Cabeza: es la superficie de rodadura que sufre el desgaste.
   - Alma: une el patín con la cabeza.
   - Patín: es el ala inferior del perfil, debe tener la anchura suficiente para que la superficie de apoyo sobre la traviesa sea grande y haya un mejor reparto de presiones.

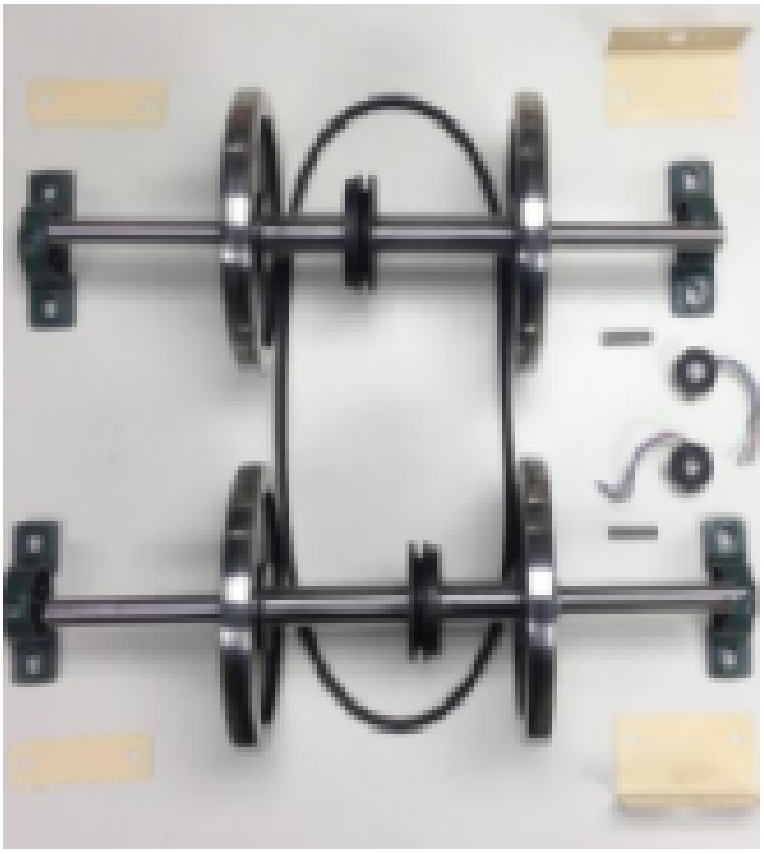

Figura 2.5: Rail

Para nuestro caso particular, en el banco de pruebas, estará una rueda vía, la cual, será una rueda que estará compuesta de un perfil igual al de una vía de tren y se encargará de simular y transmitir las fuerzas que vienen del motor, estas serán diferentes al caso real en donde la vía es inmóvil y no transmite fuerzas y las ruedas transmiten el par proveniente del motor.

Ancho de Vía.

El ancho de vía de un ferrocarril es la mínima distancia entre la parte interna de las cabezas de los perfiles, medida 14 mm por debajo del punto más alto del perfil. En las curvas se suele añadir unos milímetros más para facilitar el paso de las ruedas, el ancho de vía en el modelo sera de 0.2185 m.

2.2.1. Perfil de la Rueda

Con el objetivo de hallar el punto de contacto es importante parametrizar las curvas a las que se referencia el modelo, por lo tanto una parte importante al hallar el punto de contacto es poder parametrizar el perfil. Una manera de hacer esto es llevándolo a un conjunto de splines de orden 3 con lo cual podemos modelar las superficies de contacto como un polinomio de tercer orden. los perfiles tendrán unos coeficientes que se irán modificando, según sea el lugar del perfil en el que se encuentre. Estos son actualizados dependiendo de la posición de el punto de contacto y es aplicado antes de cada iteración usada para resolver el problema geométrico.

El perfil de la rueda estará dado de la siguiente manera :

los datos del perfil vendrán dados en la siguiente imagen:

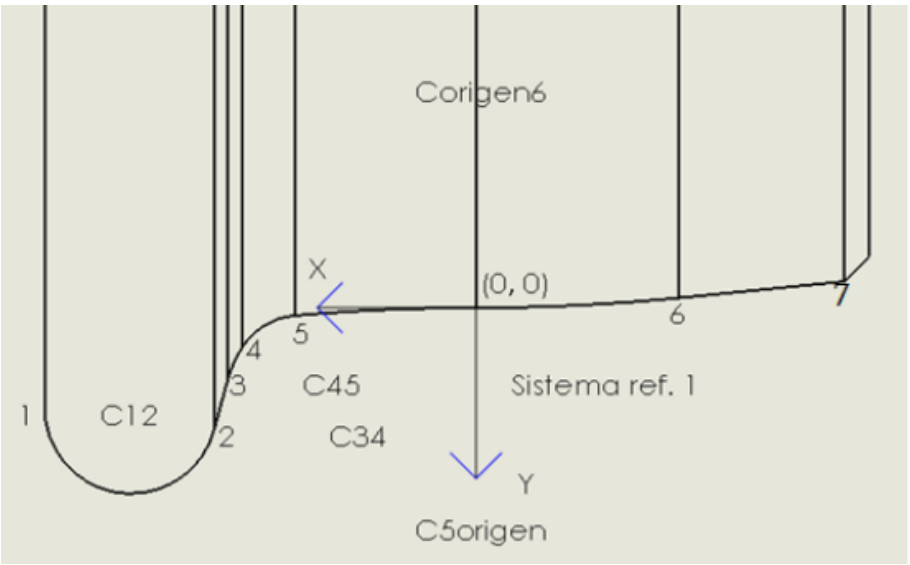

Figura 2.6: Perfil de la rueda

punto	x	y
1	22.66667	5.25
2	13.74028	6.369
3	13.07625	3.8908
4	12.31877	2.1373
5	9.51724	0.4125
origen	0	0
6	-10.66667	-0.51839
Centro 12	18.1333	5.25
Centro 34	6.7333	5.5904
Centro 45	9.2	4.0654
Centro 5origen	0	110
Centro origen6	0	-110

Figura 2.7: Datos del perfil

2.2.2. Obtención de los coeficientes de los splines

En primer lugar es importante definir el termino de spline, el cual no es mas que una una curva diferenciable definida en porciones mediante polinomios. Un spline de tercer orden es seccionar a trozos una función con funciones cúbicas o de la forma:

$$P(x) = ax^3 + bx^2 + cx + d \tag{2.1}$$

en donde se tendrán 4 variables por cada intervalo y ciertas condiciones para cada punto en común a dos intervalos, respecto de la derivada segunda:

1. Las dos funciones que pasen por el punto a aproximar sean igual a la función original en cada uno de sus puntos.

2. Que la derivada en un punto siempre coincida para ambos "lados" de la función definida a trozos que pasa por tal punto común.

3. Que la derivada segunda en un punto siempre coincida para ambos lados de la función definida a trozos que pasa por tal punto común.

Las demás ecuaciones que resuelven este sistema de ecuaciones y generan los coeficientes del spline determinan el tipo de spline que estamos considerando, sin embargo, todo este proceso es hecho en MATLAB automáticamente mediante el comando spline.

La determinación de los puntos de corte y de los coeficientes que regirán el spline de tercer orden es obtenida mediante la minimización de la norma mínima entre la función original y el spline.

En otras palabras, el procedimiento utilizado para esto, fue partir del comando de MATLAB "fminunc" el cual tiene como función hallar el valor mínimo local de una función. Con lo que partiendo de unos valores de cortes iniciales (valores a lo largo del eje x que consideran el inicio y el final del spline en cada sección), procedemos a evaluarlos en nuestra función original. (Para este caso denominaremos función original a las curvas obtenidas de los perfiles creados a partir del conjunto de datos), consiguiendo en si mismo los valores en y de la función original.

Con estos valores creamos una función a trozos de polinomios de tercer orden (splines) con el comando spline de MATLAB, esto generara una curva que pasará por lo puntos dados pero que no necesariamente tenga la forma de nuestra función original. Con lo cual nuestro objetivo en este caso es evaluar nuestra función original en la mayor cantidad de intervalos posibles y después evaluar nuestros splines en estos mismos intervalos de tal manera que al restar estos valores evaluados nos generara un vector al que denominaremos vector de error y con el cual al obtener su norma obtendremos un valor que denominaremos valor de error .

La función "fminunc" no hará mas que interpolar con un conjunto de valores de x para hallar el mínimo valor de nuestro valor de error. Lo que hará que

nuestra curva generada en los splines sea lo más parecida posible a nuestra función original, dando los coeficientes y puntos de cortes óptimos.

Con este algoritmo en sí, solo logras hacer que se obtenga la mejor distribución de puntos de cortes posible, sin embargo no es capaz por sí solo de obtener la menor cantidad de puntos de cortes. Esto fue logrado a partir de una serie de pruebas, (probando distintas cantidades de puntos de cortes , hallando los splines y comparándolos con nuestra función original), logramos establecer el número mínimo de puntos de cortes para poder representar en splines la función original o el perfil.

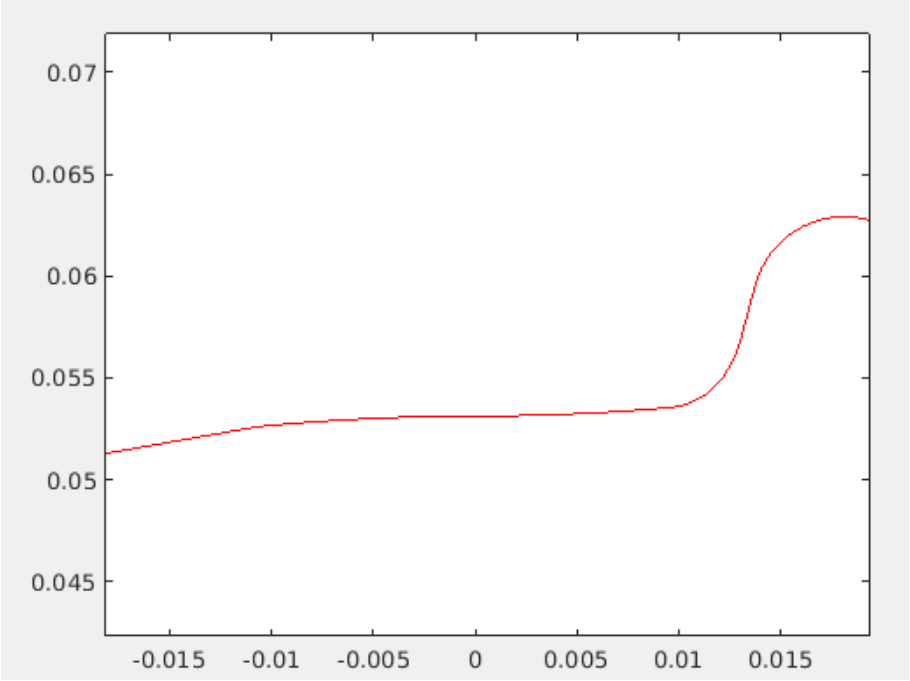

Figura 2.8: Perfil en MATLAB

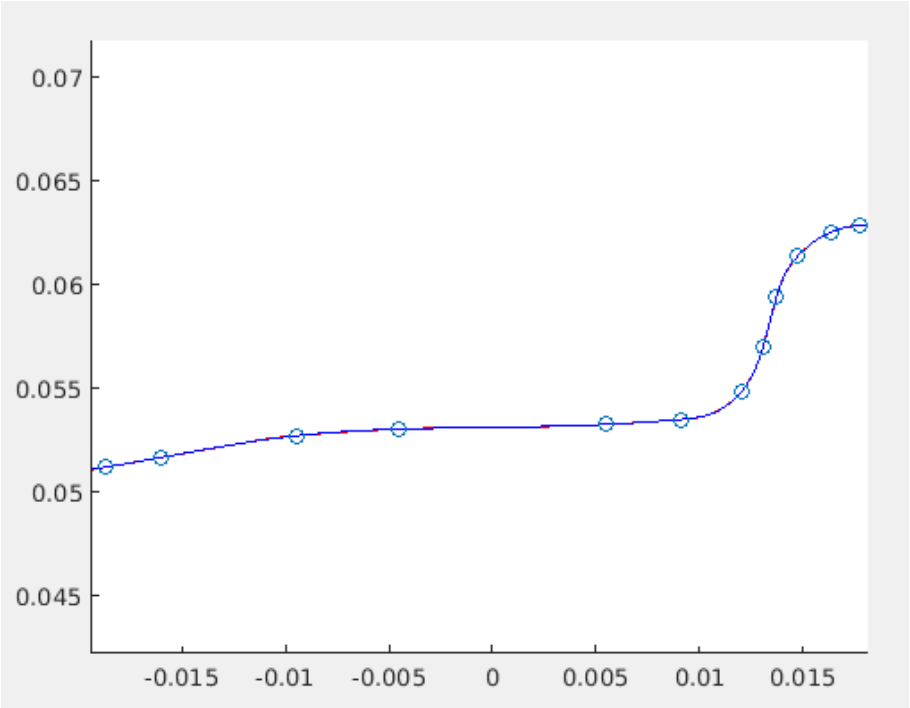

Figura 2.9: Perfil en MATLAB con splines

2.3. lib_3D_mec

El software lib_3D_mec es una librería de software libre diseñado por la UPNA cuyo propósito general es presentar una librería para resolver la dinámica multicuerpo, la cual ofrece diferentes operadores dinámicos y un lenguaje que se asemeja mucho a la mecánica tales como : vectores, tensores, puntos ,bases, matrices, y con el cual se pueden resolver y exportar las distintas matrices en distintos formatos para su posterior evaluación numérica.

Capítulo 3

Desarrollo Del Modelo

Como bien se definió anteriormente el sistema de ecuaciones a resolver nos queda de la forma:

$$\begin{bmatrix} \boldsymbol{M_{V\dot{q}}} & \boldsymbol{\Phi_q^T} \\ \boldsymbol{\Phi_{\dot{q}}} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\ddot{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\delta} \\ \boldsymbol{\gamma} \end{bmatrix} \tag{3.1}$$

En donde una vez se tenga planteado el modelo en lib_3D_mec exportaremos las matrices y funciones a utilizar para operar con Matlab a nivel numérico.

3.1. Modelo Computacional

3.1.1. Parametrización

El modelo dinámico estará compuesto por 5 solidos los cuales se denominarán:

- RAILB: eje de ruedas del raíl trasero.
- RAILF: eje de ruedas del raíl delantero.
- BOG: bogie.
- WHSB: eje de ruedas traseras del Bogie.
- WHSF: eje de ruedas delanteras del Bogie.

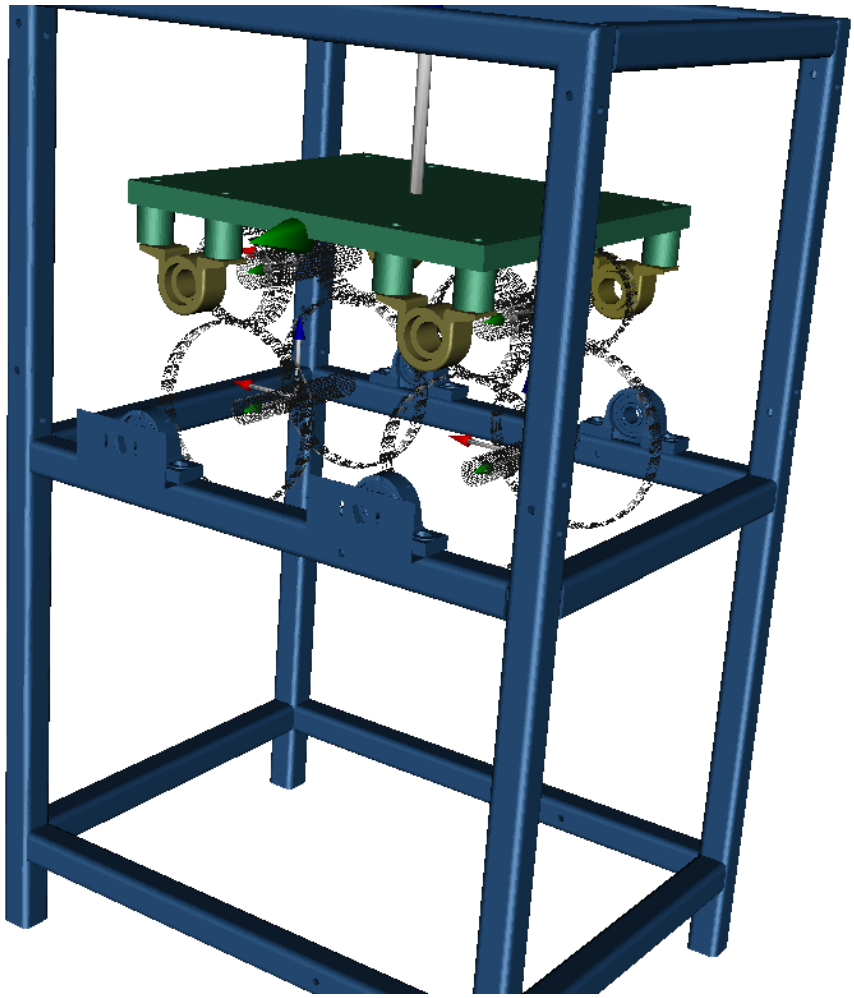

Figura 3.1: Modelo del Bogie

Figure 3.1 representa la Parametrización del Bogie.

Definición de las Bases:

$$xyz \xrightarrow{(\mathbf{e}^{xyz}, bRailB)} B\_RailB, \tag{3.2}$$

$$xyz \xrightarrow{(\mathbf{e}^{xyz}, bRailF)} B\_RailF, \tag{3.3}$$

$$xyz \xrightarrow{(\mathbf{e}^{xyz}, cCh)} B\_Ch\_c, \tag{3.4}$$

$$B\_Ch\_c \xrightarrow{(\mathbf{e}^{B\_Ch\_c}, aCh)} B\_Ch\_a, \tag{3.5}$$

$$B\_Ch\_a \xrightarrow{(\mathbf{e}^{B\_Ch\_a}, bCh)} B\_Ch, \tag{3.6}$$

$$xyz \xrightarrow{(\mathbf{e}^{xyz}, cWhsB)} B\_WhsB\_c, \tag{3.7}$$

$$B\_WhsB\_c \xrightarrow{(\mathbf{e}^{B\_WhsB\_c}, aWhsB)} B\_WhsB\_a, \tag{3.8}$$

$$B\_WhsB\_a \xrightarrow{(\mathbf{e}^{B\_WhsB\_a}, bWhsB)} B\_WhsB, \tag{3.9}$$

$$xyz \xrightarrow{(\mathbf{e}^{xyz}, cWhsF)} B\_WhsF\_c, \tag{3.10}$$

$$B\_WhsF\_c \xrightarrow{(\mathbf{e}^{B\_WhsF\_c}, aWhsF)} B\_WhsF\_a, \tag{3.11}$$

$$B\_WhsF\_a \xrightarrow{(\mathbf{e}^{B\_WhsF\_a}, bWhsF)} B\_WhsF, \tag{3.12}$$

El diagrama anterior muestra esquemáticamente la definición de las diferentes bases. Por ejemplo, la base B_RailB es la obtenida al girar un ángulo bRailB alrededor eje Y de la base xyz que otorga la orientación del cuerpo inercial.

Definición de puntos:

$$O \xrightarrow{-0{,}15625\ \mathbf{e}_x} O\_RailB, \tag{3.13}$$

$$O \xrightarrow{0{,}15625\ \mathbf{e}_x} O\_RailF, \tag{3.14}$$

$$O \xrightarrow{xCh\ \mathbf{e}_x + yCh\ \mathbf{e}_y + zCh\ \mathbf{e}_z} O\_Ch, \tag{3.15}$$

$$O\_Ch \xrightarrow{xWhsB\ \mathbf{e}_x WhsB + yWhsB\ \mathbf{e}_y WhsB + zWhsB\ \mathbf{e}_z WhsB} O\_WhsB, \tag{3.16}$$

$$O\_Ch \xrightarrow{xWhsF\ \mathbf{e}_x WhsF + yWhsF\ \mathbf{e}_y WhsF + zWhsF\ \mathbf{e}_z WhsF} O\_WhsF, \tag{3.17}$$

$$O\_WhsB \xrightarrow{0{,}188\ \mathbf{e}_y WhsB} C\_BearBL\_WhsB, \tag{3.18}$$

$$O\_WhsB \xrightarrow{-0{,}188\ \mathbf{e}_y WhsB} C\_BearBR\_WhsB, \tag{3.19}$$

$$O\_WhsF \xrightarrow{0{,}188\ \mathbf{e}_y WhsF} C\_BearFL\_WhsF, \tag{3.20}$$

$$O\_WhsF \xrightarrow{-0{,}188\ \mathbf{e}_y WhsF} C\_BearFR\_WhsF, \tag{3.21}$$

$$O \xrightarrow{-0{,}15625\ \mathbf{e}_x + 0{,}188\ \mathbf{e}_y + -0{,}0775\ \mathbf{e}_z} C\_BearBL\_Ch, \tag{3.22}$$

$$O \xrightarrow{-0{,}15625\ \mathbf{e}_x - 0{,}188\ \mathbf{e}_y + -0{,}0775\ \mathbf{e}_z} C\_BearBR\_Ch, \tag{3.23}$$

$$O \xrightarrow{0{,}15625\ \mathbf{e}_x + 0{,}188\ \mathbf{e}_y + -0{,}0775\ \mathbf{e}_z} C\_BearFL\_Ch, \tag{3.24}$$

$$O \xrightarrow{0{,}15625\ \mathbf{e}_x - 0{,}188\ \mathbf{e}_y + -0{,}0775\ \mathbf{e}_z} C\_BearFR\_Ch, \tag{3.25}$$

De forma similar, el anterior diagrama muestra esquemáticamente la definición de los diferentes puntos del modelo. Por ejemplo el punto C_BearFR_Ch es el punto en el centro de los cojinetes el cual esta ubicado a una distancia de 0.15625 en la dirección del eje x , 0.0188 en la dirección del eje y, -0.0775 en la dirección del eje z. Los vectores de las coordenadas generalizadas, velocidades y aceleraciones quedarán de la siguiente manera:

$$\mathbf{q} = \begin{bmatrix} bRailB \\ bRailF \\ xCh \\ yCh \\ zCh \\ cCh \\ aCh \\ bCh \\ bWhsB \\ cWhsB \\ aWhsB \\ xWhsB \\ yWhsB \\ cWhsB \\ bWhsF \\ cWhsF \\ aWhsF \\ xWhsF \\ yWhsF \\ zWhsF \\ thetaW_BL \\ uW_BL \\ uR_BL \\ thetaR_BL \\ thetaW_BR \\ uW_BR \\ uR_BR \\ thetaR_BR \\ thetaW_FL \\ uW_FL \\ uR_FL \\ thetaR_FL \\ thetaW_FR \\ uW_FR \\ uR_FR \\ thetaR_FR, \end{bmatrix} \quad \dot{\mathbf{q}} = \begin{bmatrix} \dot{bRailB} \\ \dot{bRailF} \\ \dot{xCh} \\ \dot{yCh} \\ \dot{zCh} \\ \dot{cCh} \\ \dot{aCh} \\ \dot{bCh} \\ \dot{bWhsB} \\ \dot{cWhsB} \\ \dot{aWhsB} \\ \dot{xWhsB} \\ \dot{yWhsB} \\ \dot{cWhsB} \\ \dot{bWhsF} \\ \dot{cWhsF} \\ \dot{aWhsF} \\ \dot{xWhsF} \\ \dot{yWhsF} \\ \dot{zWhsF} \\ \dot{thetaW}_BL \\ \dot{uW}_BL \\ \dot{uR}_BL \\ \dot{thetaR}_BL \\ \dot{thetaW}_BR \\ \dot{uW}_BR \\ \dot{uR}_BR \\ \dot{thetaR}_BR \\ \dot{thetaW}_FL \\ \dot{uW}_FL \\ \dot{uR}_FL \\ \dot{thetaR}_FL \\ \dot{thetaW}_FR \\ \dot{uW}_FR \\ \dot{uR}_FR \\ \dot{thetaR}_FR, \end{bmatrix} \quad \ddot{\mathbf{q}} = \begin{bmatrix} \ddot{bRailB} \\ \ddot{bRailF} \\ \ddot{xCh} \\ \ddot{yCh} \\ \ddot{zCh} \\ \ddot{cCh} \\ \ddot{aCh} \\ \ddot{bCh} \\ \ddot{bWhsB} \\ \ddot{cWhsB} \\ \ddot{aWhsB} \\ \ddot{xWhsB} \\ \ddot{yWhsB} \\ \ddot{cWhsB} \\ \ddot{bWhsF} \\ \ddot{cWhsF} \\ \ddot{aWhsF} \\ \ddot{xWhsF} \\ \ddot{yWhsF} \\ \ddot{zWhsF} \\ \ddot{thetaW}_BL \\ \ddot{uW}_BL \\ \ddot{uR}_BL \\ \ddot{thetaR}_BL \\ \ddot{thetaW}_BR \\ \ddot{uW}_BR \\ \ddot{uR}_BR \\ \ddot{thetaR}_BR \\ \ddot{thetaW}_FL \\ \ddot{uW}_FL \\ \ddot{uR}_FL \\ \ddot{thetaR}_FL \\ \ddot{thetaW}_FR \\ \ddot{uW}_FR \\ \ddot{uR}_FR \\ \ddot{thetaR}_FR, \end{bmatrix} \tag{3.26}$$

3.1.2. Restricciones

En general se tiene que cada una de las ruedas del bogie presentará la mismas ecuaciones de restricción. Cada rueda a su vez estará compuesta de 5 restricciones en donde restringiremos los vectores tangentes, y los vectores normales del raíl con los de la rueda en el punto de contacto J. Estas nos darán las restricciones suficientes para que se tenga el mismo punto de contacto. En este caso obligamos a que los vectores tangentes en el punto de contacto sean perpendiculares al vector normal. Por lo que las ecuaciones nos quedan de la siguiente manera:

$$\Phi = \begin{bmatrix} \mathbf{t}_1^r \cdot \mathbf{r}_{Pr}^{Pw} = 0 \\ \mathbf{t}_2^r \cdot \mathbf{r}_{Pr}^{Pw} = 0 \\ \mathbf{t}_1^w \cdot \mathbf{n}^r = 0 \\ \mathbf{t}_2^w \cdot \mathbf{n}^r = 0 \\ \mathbf{n}^r \cdot \mathbf{r}_{Pr}^{Pw} = 0 \end{bmatrix} \tag{3.27}$$

Donde definimos como 1 y 2 como las direcciones x e y del sistema absoluto de referencia, también tenemos que $\mathbf{r}_{Pr}^{Pw}$ representan los puntos de contactos entre las superficies de la rueda y el raíl respectivamente. Los vectores tangentes y normal de la rueda y el raíl definidos en el punto de contacto están definidos de la siguiente manera:

$$\mathbf{t}_1^r = \frac{\frac{\partial \mathbf{r}_r^{P^r}}{\partial s}}{|\frac{\partial \mathbf{r}_r^{Pr}}{\partial s}|}, \tag{3.28}$$

$$\mathbf{t}_2^r = \frac{\frac{\partial_{\mathbf{r}}{}^{Pw_r}}{\partial_{U_r}}}{|\frac{\partial_{\mathbf{r}}{}^{Pw_r}}{\partial_{U_r}}|}, \tag{3.29}$$

$$\mathbf{n}^r = \mathbf{t}_x^r \times \mathbf{t}_y^r \tag{3.30}$$

$$\mathbf{t}_1^w = \frac{\frac{\partial_{\mathbf{r}}{}^{Pw_r}}{\partial_\alpha}}{|\frac{\partial_{\mathbf{r}}{}^{Pw_r}}{\partial_\alpha}|}, \tag{3.31}$$

$$\mathbf{t}_2^w = \frac{\frac{\partial_{\mathbf{r}}{}^{Pw_r}}{\partial_{U_w}}}{|\frac{\partial_{\mathbf{r}}{}^{Pw_r}}{\partial_{U_w}}|} \tag{3.32}$$

Donde expresamos como la posición del punto de contacto $\mathbf{P}^r$ medido a lo largo del raíl, y $\boldsymbol{U}$r como la posición del punto de contacto medido en la dirección perpendicular al raíl.

3.2. Nomenclatura

Al ser un sistema formado por diferentes elementos es importante saber cómo se han definido las diferentes coordenadas y parámetros. También es importante tener una nomenclatura que permita saber rápidamente a qué nos referimos. por

ejemplo en nuestro caso el punto de origen (o punto inicial donde nos empezaremos a tomar como referencia el resto del modelo), lo llamamos punto O y se situara en el [0,0,0] de la referencia absoluta.

3.2.1. Sólidos

Para hacer referencia a los sólidos es necesaria una notación corta pero que sea clara, para establecer el tipo de elemento al que nos referimos.

La notación, que usaremos para los sólidos se expresara de la siguiente manera: se usará una abreviatura del nombre a usar, para nuestro caso como tenemos 5 sólidos quedarán puestos de la siguiente manera:

- El Wheelset trasero (ruedas del tren) quedará: WhsB.
- El Wheelset delantero quedará: WhsF.
- El Chásis del tren quedará: Ch.
- El Raíl delantero quedará: RailF.
- El Raíl trasero quedará: RailB.

Tenemos en cuenta que para la notación hemos decidido en poner en mayúscula la primera letra del solido, abreviar el nombre y definir su ubicación con respecto al espacio mediante una letra en mayúscula donde definimos por defecto B como parte trasera (Back) y F como parte delantera (Front). En otros apartados de otras definiciones tales como los cojinetes se tiene en cuenta el hecho de que se puedan encontrar a la izquierda o a la derecha del centro de referencia inicial. Por lo que un factor muy importante a tomar en cuenta es si queda a la izquierda o a la derecha del centro para este caso continuaremos con la notación y le daremos como nombre la primera letra en mayúscula, y las ultimas 2 del nombre en mayúscula indicando la ubicación por ejemplo (C_BearBL).

3.2.2. Bases

Para el caso de las bases, estas las hemos definido en el código poniendo en primer lugar una $B\_$ y después agregamos el nombre del sólido a la cual estuviese asociada esta base, en caso de base intermedia al final del nombre damos cuenta del ángulo que se gira para obtener la base. Por ejemplo si en nuestro caso tenemos B_Ch_c,, donde vemos que $B\_$ indica que estamos hablando de una base, Ch que indica que estamos hablando de un sólido y c indica mediante cual ángulo fue girada.

3.2.3. Puntos

En general se tienen diferentes puntos a los cuales se tendrán asociados ciertas funciones.

- Para los puntos que expresan el Origen de un sólido lo denotamos como $O\_$ y agregamos el nombre del sólido.
- Para los centros de solidos usamos como nomenclatura $C\_$ y agregamos el nombre del sólido y después el solido al que pertenece.
- Para definir el punto de contacto lo denotamos como el punto $J\_$ y nombramos lugar y después a la superficie que pertenece(ejemplo:J_BR_Wheel donde en primer lugar ponemos $J\_$, después la ubicación BR y después a que superficie en este caso la rueda (wheel).

Capítulo 4

El Modelo lineal de Kalker

4.1. Modelo lineal de Kalker

Una de las partes mas importantes a tomar en cuenta en el modelo era la interacción de fuerzas entre la rueda y el raíl. Para determinar estas fuerzas lo primero que hay que hacer es hallar algunos parámetros de contacto, tales como: la superficie de contacto, la presión y las fuerzas tangenciales. Para hallar estos parámetros se suelen resolver los siguientes casos:

- El problema normal (teoría de Hertz).
- El problema tangencial(teoría de Kalker).

4.1.1. El Problema Normal

El estudio del contacto entre cuerpos es posible hoy día con métodos de elementos finitos. Sin embargo, la necesidad para calcular lo mas rápido posible en los códigos dinámicos nos lleva al uso de métodos analíticos.

El Contacto de Hertz

Hertz demostró que cuando dos cuerpos elásticos son presionados entre ellos bajo las siguientes condiciones:

- Comportamiento elástico.
- Espacios semi-infinitos.
- Largos radios de curvatura comparados al tamaño del contacto.
- Curvaturas constantes dentro del área de contacto.

Entonces:

- La superficie de contacto es una elipse.

- La superficie de contacto es considerada plana.
- La presión tiene la forma de semi-elipse.

La curvaturas principales de los dos semi espacios son necesarias para los cálculos de la dimensiones de la superficie y su distribución de presión. En el caso de las vías de un tren, las 4 curvaturas principales se encuentran en planos perpendiculares; sus direcciones corresponden a los ejes principales de la estructura.

Considerando los dos cuerpos elásticos, estos se encontrarán en un punto donde la distancia entre ellos es mínima. Cerca de este punto de contacto, sin carga alguna, las formas de las superficies de los cuerpos están representadas por dos polinomios de segundo orden:

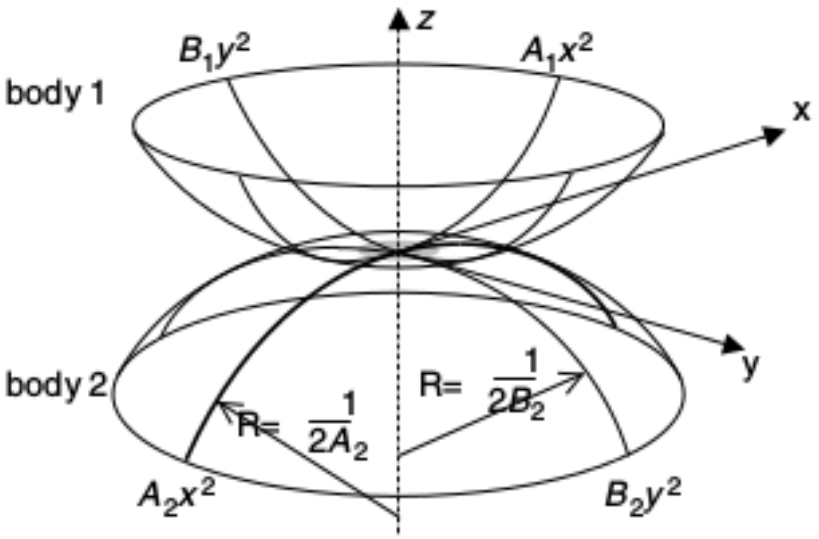

Figura 4.1: Contacto de Hertz

$$z_1 = A_1\mathbf{x}^2 + B_1\mathbf{y}^2 \tag{4.1}$$
$$z_2 = A_2\mathbf{x}^2 + B_2\mathbf{y}^2 \tag{4.2}$$

Los coeficientes $A_{1,2}$ y $B_{1,2}$ se asumen constantes cercanos al punto de contacto. y son obtenidos mediante el desarrollo de la serie de Taylor donde despreciaremos su contribución de primer orden por lo que tendríamos una solución de la siguiente manera:

$$\frac{\partial^2 z_1}{\partial_x^2} = 2A_1 \approx \frac{1}{r_{wn}} \tag{4.3}$$

$$\frac{\partial^2 z_1}{\partial_y^2} = 2B_1 \approx \frac{1}{R_{wx}} \tag{4.4}$$

$$\frac{\partial^2 z_2}{\partial_x^2} = 2A_2 \approx \frac{1}{r_{rn}} \tag{4.5}$$

$$\frac{\partial^2 z_2}{\partial_y^2} = 2B_2 \approx \frac{1}{R_{rx}} \tag{4.6}$$

El coeficiente A_2 es usualmente despreciado debido a que los rieles tienden a ser rectos, sin embargo para nuestro caso particular, esto no solo existirá, sino que tendrá que ser tomado en cuenta en las ecuaciones para el modelo.

Relación A/B con los Semiejes b/a

Antes de ser sometida a una carga, la distancia vertical relativa entre los dos cuerpos puede ser escrita de la siguiente manera:

$$z_1 + z_2 = A\mathbf{x}^2 + B\mathbf{y}^2 \tag{4.7}$$

con:

$$A = \frac{1}{2r_{wn}} + \frac{1}{2r_{rn}} \tag{4.8}$$

$$B = \frac{1}{2R_{wx}} + \frac{1}{2R_{rx}} \tag{4.9}$$

donde A y B tendrán que ser estrictamente positivos.

Convencionalmente, a es el semieje de la elipse en la dirección x y b es el semieje de la elipse en la dirección transversal. La relación de A/B y b/a variaran de la misma manera por lo que si $A > B$, entonces $b > a$ la igualdad nos llevara un área de contacto circular.

Cálculo de los Semiejes

El cálculo tradicional esta basado en hallar la relación de los semiejes: $g < 1$,$(g = b/a)$ ó $(g = a/b)$, en función de A y B usando un parámetro intermedio que llamaremos θ definido como:

$$cos(\theta) = \frac{|B - A|}{B + A} \tag{4.10}$$

Los valores prácticos de los semiejes a y b, y θ como la reducción de la distancia entre cuerpos centrados estará dada por:

Si $a > b$

$$a = m\left(\frac{3}{2}N\frac{1-\upsilon^2}{E}\frac{1}{A+B}\right)^{1/3} \tag{4.11}$$

$$b = n\left(\frac{3}{2}N\frac{1-\upsilon^2}{E}\frac{1}{A+B}\right)^{1/3} \tag{4.12}$$

$$\delta = r\left(\left(\frac{3}{2}N\frac{1-\upsilon^2}{E}\right)^2(A+B)\right)^{1/3} \tag{4.13}$$

Siendo E el módulo de Young y υ el modulo de Poisson, suponemos que el material del raíl y la rueda es el mismo.

m,n y r son coeficientes adimensionales tabulados en función de la relación $g = n/m$ o del ángulo θ.

por lo que si πab es el perímetro de la elipse, esta puede ser expresada en función de:

$$ab = mn\left(\frac{3}{2}N\frac{1-\upsilon^2}{E}\frac{1}{A+B}\right)^{2/3}N^{2/3} \tag{4.14}$$

en donde el primer término contiene el material y las constantes geométricas, y el segundo sólo la carga.

Presión de Contacto para el Modelo

Con una distribución de presión elíptica, las presiones principales serian N/πab, por lo que la máxima presión podrá ser simplificada como:

$$\sigma_{max} = 1{,}5N/\pi \tag{4.15}$$

En el campo de los ferrocarriles, la presión máxima de contacto esta frecuentemente sobre los 1000 Mpa. Este valor está por encima del límite elástico de la mayoría de los aceros, sin embargo el estado de compresión de los materiales, es mucho mas complejo que un simple ensayo a tensión por lo que el límite elástico no es alcanzado. La determinación de la plastificación (el cual es el límite de la hipótesis de Hertz) debe ser calculado con un criterio basado en los esfuerzos hidrostáticos (Von Mises).

Hertz Coefficients (A/B < 1)

$\theta°$	90	80	70	60	50	40	30	20	10	0
$g = n/m$	1	0.7916	0.6225	0.4828	0.3652	0.2656	0.1806	0.1080	0.0470	0
m	1	1.128	1.285	1.486	1.754	2.136	2.731	3.816	6.612	∞
n	1	0.8927	0.8000	0.7171	0.6407	0.5673	0.4931	0.4122	0.3110	0
r	1	0.9932	0.9726	0.9376	0.8867	0.8177	0.7263	0.6038	0.4280	0

Figura 4.2: coeficientes de Hertz

Hertz Coefficients for $\theta = 0$ to 180°

$\theta°$	0	5	10	30	60	90	120	150	170	175	180
A/B	0	0.0019	0.0077	0.0717	0.3333	1	3.0	13.93	130.6	524.6	∞
$b/a = n/m$	0	0.0212	0.0470	0.1806	0.4826	1	2.0720	5.5380	21.26	47.20	∞
m	∞	11.238	6.612	2.731	1.486	1	0.7171	0.4931	0.311	0.2381	0
r	0	0.2969	0.4280	0.7263	0.9376	1	0.9376	0.7263	0.4280	0.2969	0

Figura 4.3: coeficientes de Hertz cont.

TABLE 4.3
Approximation of the n/m Values

$\theta°$	0	5	10	30	60	90	120	150	170	175	180
A/B	0	0.0019	0.0077	0.0717	0.3333	1	3.0	13.93	130.6	524.6	∞
$b/a = n/m$	0	0.0212	0.0470	0.1806	0.4826	1	2.0720	5.5380	21.26	47.20	∞
$(A/B)^{0.63}$	0	0.0193	0.0466	0.1901	0.5005	1	1.9980	5.2564	21.530	51.700	∞

Figura 4.4: coeficientes de Hertz cont2.

4.1.2. El Problema Tangencial

La representación dinámica de las ruedas han sido usadas por largo tiempo para explicar el comportamiento sinusoidal de un conjunto de ruedas de tren libre. Sin embargo la situación es diferente en un vehículo real.

El eje de ruedas del tren está fuertemente unido al vehículo mediante elementos de suspensión flexibles, y estas uniones crean fuerzas significativas cuando el conjunto de ruedas está entrando a una curva o recorriendo una vía real con irregularidades.

Las fuerzas de la suspensión encuentran sus fuerzas de reacción en la interfase donde los componentes tangentes o de arrastre están relacionados a la velocidad relativa de los cuerpos:"los creepages"(o deslizamiento).

En el sistema de coordenadas en el punto de contacto, las fuerzas son llamadas de la siguiente manera:

- N para las fuerzas normales.
- F_x para las fuerzas de creep (o de deslizamiento) en el sentido longitudinal.
- F_y para las fuerzas de creep en el plano de contacto.

La fuerza Fy debe ser proyectada en el plano de la pista OY y sumado para dar la fuerzas de guía.

Fuerzas de Creep

En el caso del contacto lineal de Kalker, las fuerzas de creep (o de deslizamiento) están en función de las velocidades relativas entre los cuerpos rígidos cercanos al punto de contacto.(los creepages).

La expresión general de las fuerzas de creep tomaran en cuenta los coeficientes de rigidez expresados para en la teoría lineal de Kalker. Las cuales quedarán expresadas de la siguiente manera:

$$F_x = -Gabc_{11}\upsilon_x \tag{4.16}$$

$$F_y = -Gabc_{22}\upsilon_y - Gabc_{23}c\phi \qquad (con \quad c = \sqrt{ab}) \tag{4.17}$$

$$M_z = Gabc_{22}\upsilon_y - Gabc_{33}\phi \tag{4.18}$$

Donde G es el módulo de corte del material (acero); πab es la superficie de contacto y c_{ij} los coeficientes obtenidos por la tabla.

		C_{11}			C_{22}			$C_{23} = -C_{32}$			C_{33}		
	g	$\sigma = 0$	1/4	1/2	$\sigma = 0$	1/4	1/2	$\sigma = 0$	1/4	1/2	$\sigma = 0$	1/4	1/2
	0.0	$\pi^2/4(1-\sigma)$			$\pi^2/4 = 2{,}47$			$\pi\sqrt{g}/3$	—	—	$\pi^2/16(1-\sigma)g$		
a/b	0.1	2.51	3.31	4.85	2.51	2.52	2.53	0.334	0.473	0.731	6.42	8.28	11.7
	0.2	2.59	3.37	4.81	2.59	2.63	2.66	0.483	0.603	0.809	3.46	4.27	5.66
	0.3	2.68	3.44	4.80	2.68	2.75	2.81	0.607	0.715	0.889	2.49	2.96	3.72
	0.4	2.78	3.53	4.82	2.78	2.88	2.98	0.720	0.823	0.977	2.02	2.32	2.77
	0.5	2.88	3.62	4.83	2.88	3.01	3.14	0.827	0.929	1.07	1.74	1.93	2.22
	0.6	2.98	3.72	4.91	2.98	3.14	3.31	0.930	1.03	1.18	1.56	1.68	1.86
	0.7	3.09	3.81	4.97	3.09	3.28	3.48	1.03	1.14	1.29	1.43	1.50	1.60
	0.8	3.19	3.91	5.05	3.19	3.41	3.65	1.13	1.25	1.40	1.34	1.37	1.42
	0.9	3.29	4.01	5.12	3.29	3.54	3.82	1.23	1.36	1.51	1.27	1.27	1.27
b/a	1.0	3.40	4.12	5.20	3.40	3.67	3.98	1.33	1.47	1.63	1.21	1.19	1.16
	0.9	3.51	4.22	5.30	3.51	3.81	4.16	1.44	1.59	1.77	1.16	1.11	1.06
	0.8	3.65	4.36	5.42	3.65	3.99	4.39	1.58	1.75	1.94	1.10	1.04	0.954
	0.7	3.82	4.54	5.58	3.82	4.21	4.67	1.76	1.95	2.18	1.05	0.965	0.852
	0.6	4.06	4.78	5.80	4.06	4.50	5.04	2.01	2.23	2.50	1.01	0.892	0.751
	0.5	4.37	5.10	6.11	4.37	4.90	5.56	2.35	2.62	2.96	0.958	0.819	0.650
	0.4	4.84	5.57	5.57	4.84	5.48	6.31	2.88	3.24	3.70	0.912	0.747	0.549
	0.3	5.57	6.34	7.34	5.57	6.40	7.51	3.79	4.32	5.01	0.868	0.674	0.446
	0.2	6.96	7.78	8.82	6.96	8.14	9.79	5.72	6.63	7.89	0.828	0.601	0.341
	0.1	10.7	11.7	12.9	10.7	12.8	16.0	12.2	14.6	18.0	0.795	0.526	0.228

Figura 4.5: coeficientes de Kalker.

Definición de los Creepages

Una expresión general para dos cuerpos que giran puede ser obtenida mediante la proyección de los vectores de velocidad en x,y y z.

las expresiones quedarán de la siguiente manera:

$$-longitudinal \quad v_x = \frac{proj./x(V_0 - V_1)}{\frac{1}{2}(V_0 + V_1)} \qquad (adimensional) \tag{4.19}$$

$$-lateral \quad v_y = \frac{proj./y(V_0 - V_1)}{\frac{1}{2}(V_0 + V_1)} \qquad (adimensional) \tag{4.20}$$

$$-Rotacional \quad \phi = \frac{proj./z(\omega_0 - \omega_1)}{\frac{1}{2}(V_0 + V_1)} \qquad (1/m) \tag{4.21}$$

Las expresiones ω_0 y ω_1 representan la velocidad angular de dos solidos proyectadas en el vector normal de contacto. Donde las velocidades V_0 y V_1 son las velocidades absolutas en el punto de contacto por lo cual la expresión $1/2(V_0 + V_1)$ sera la velocidad promedio.

Capítulo 5

Implementación de Kalker en el Modelo Multicuerpo

Como podemos ver para el modelo de Kalker se necesitan diferentes variables de entrada, que debemos proporcionar para poder obtener como salida las fuerzas de Kalker. Teniendo en cuenta la variación del estado en el sistema una de las metas es poder calcular con exactitud y rapidez los estados siguientes con el fin de obtener una simulación del comportamiento del sistema de manera rápida.

5.1. Parametrización de la superficie

La superficies de las ruedas se supone que deben de tener una simetría cilíndrica, al igual que los rieles, el perfil del raíl y las ruedas son paramentados usando splines de orden 3 . Las superficies de contacto son modeladas como un polinomio simple de tercer orden que representan los perfiles, los cuales tendrán unos coeficientes que irán modificándose según sea el lugar del perfil en el que se encuentre. Estos son actualizados dependiendo de la posición del punto de contacto y es aplicado antes de cada iteración usada para resolver el problema geométrico. Los polinomios quedan expresados de la siguiente manera:

$$sW = (uW - lW) \tag{5.1}$$

$$SplineW = dW + sW(cW + sW(bW + sW * aW)) \tag{5.2}$$

De igual forma para el Rail

$$sR = (uR - lR) \tag{5.3}$$

$$SplineR = dR + sR(cR + sR(bR + sR * aR)) \tag{5.4}$$

donde los coeficientes a,b,c,d y los puntos de corte l son los valores actualizados que dependerán del punto de contacto. u será la coordenada generalizada para dar la posición del punto de contacto en los perfiles.

En las coordenadas generalizadas θ es la usada para dar el punto de contacto (J) en la posición angular.

Para actualizar los coeficientes del spline se tiene una función especifica en MATLAB la cual partiendo del valor U hallamos en que sección de la la spline nos encontramos y el valor en y respectivo a la sección del perfil(no es mas que evaluar nuestra función del perfil y obtener en que lugar exacto nos encontramos al punto de contacto).

5.2. Fuerzas Externas en el Modelo

5.2.1. Muelles de Unión

Además de las fuerzas de gravedad, se implementaron diferentes fuerzas de enlace en el modelo, con el fin de poder simular uniones muelle-amortiguadas entre los sólidos. Por lo que se implementaron una fuerza de acción y reacción en un punto ubicado en el centro del rodamiento definido en el Wheelset con un punto que estará definido desde el chasis e irá al centro del rodamiento, en donde entre estos 2 puntos existirán una fuerza de acción y una fuerza de reacción, que harán que se junten los cuerpos.

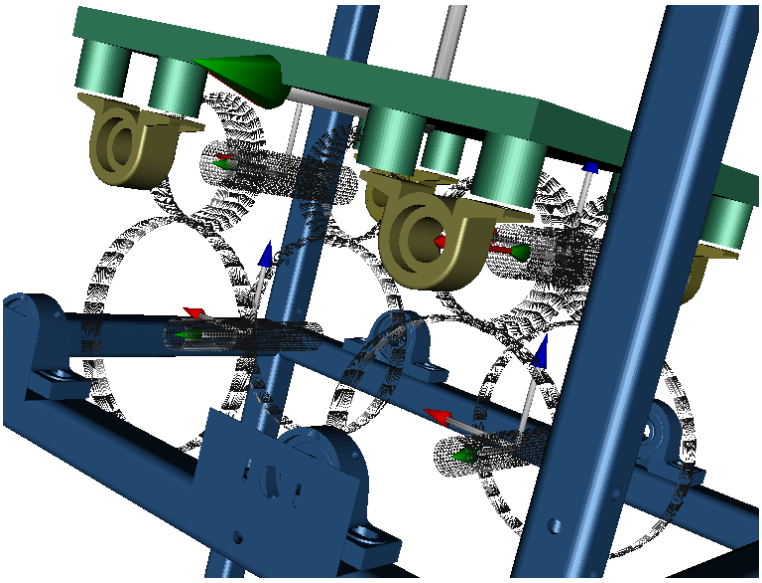

Figura 5.1: vector de posición del centro del rodamiento definido desde el Wheelset.

En la 5.1 podemos ver la definición dada por el punto. De igual manera se definió con respecto al chasis como se muestra a continuación:

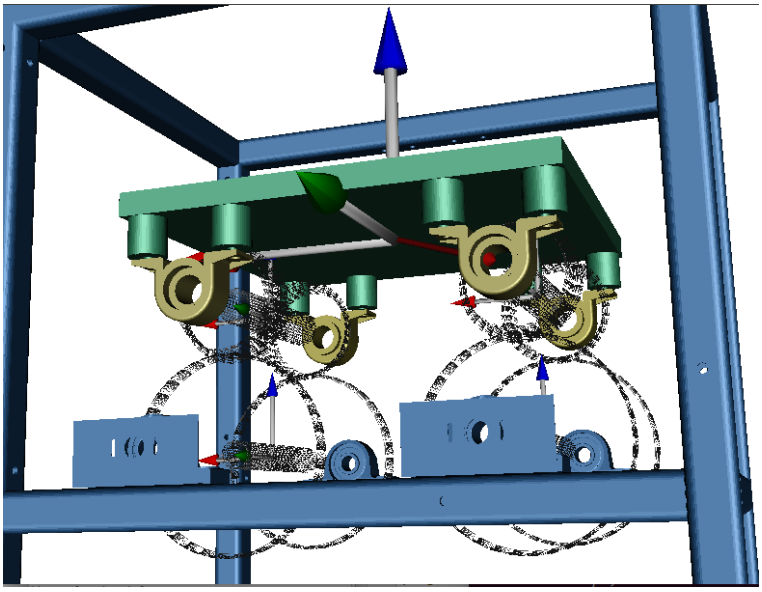

Figura 5.2: vector de posición del centro del rodamiento definido desde el chasis

En la figura 5.2 es una muestra de como definimos el punto del rodamiento pero esta vez desde el chasis.

5.2.2. Muelle de fijación

Otro de los temas que teníamos que lograr simular era el hecho de que el chasis está suspendido con una masa la cual evita el movimiento parcial en la dirección de x manteniendo el Bogie dentro de la estructura. Razón por la cual unas fuerzas de enlaces fueron implementadas de manera de que el desplazamiento en la dirección x de la estructura tendrán una fuerza de reacción en opuesta al movimiento, la cual disminuirá de manera parcial o total el movimiento en esta dirección, siendo esto ultimo lo buscado.

5.2.3. Fuerzas de Kalker

Este es uno de los parámetros más complicados, puesto que su solución tenia que darse en MATLAB sin embargo es parte del modelo. Mas adelante veremos el procedimiento para la obtención de estas fuerzas. Sin embargo los cálculos aplicados fueron basados en la teoría de Hertz, hay que tener en cuenta que estas fuerzas son variables de entrada al modelo, las cuales para tendremos que estar actualizando para que podamos resolver las coordenadas en un instante de tiempo dado.

5.2.4. Par Torsor en el Raíl

Una de las condiciones a reproducir en el modelo es el del par transmitido por un motor el cual dará el movimiento al sistema. Como input al sistema se tiene un par torsor el cual dará el movimiento a cada raíl.

5.3. Obtención de las Variables Mediante el Modelo Dinámico

Para obtener las variables de entrada de manera rápida, y como las fuerza normal es obtenida a partir del modelo dinámico, (ya que en este tomamos en cuenta las fuerzas de enlace), tomamos como premisa lo siguiente:

- Que al tener un paso de integración corto, las fuerzas obtenidas en el instante anterior son muy próximas o muy cercanas al estado siguiente.
- Que las fuerzas de enlace obtenidas en el modelo dinámico serán iguales a la fuerza normal.

5.3.1. Obtención de las Velocidades de Creep en el Modelo

Para hallar los creepages en nuestro modelo, usamos la siguientes ecuaciones:

$$\boldsymbol{v_x} = \frac{V_{P^w_w}^{P^w_w} - V_{P^r_r}^{P^r_r}}{\frac{1}{2}|V_{P^w_w}^{P^w_w} + V_{P^r_r}^{P^r_r}|}\boldsymbol{t_x^r} \tag{5.5}$$

$$\boldsymbol{v_y} = \frac{V_{P^w_w}^{P^w_w} - V_{P^r_r}^{P^r_r}}{\frac{1}{2}|V_{P^w_w}^{P^w_w} + V_{P^r_r}^{P^r_r}|}\boldsymbol{t_y^r} \tag{5.6}$$

$$\boldsymbol{\phi} = \frac{\omega_{P^w_w}^{P^w_w} - \omega_{P^r_r}^{P^r_r}}{\frac{1}{2}|V_{P^w_w}^{P^w_w} + V_{P^r_r}^{P^r_r}|}\boldsymbol{n^r} \tag{5.7}$$

donde P_w^w y P_r^r, representa la velocidad lineal en el punto de contacto respecto de la rueda y con respecto al raíl respectivamente.

5.4. Obtención de de las fuerzas de Kalker

A partir de los parámetros geométricos dados anteriormente, las fuerzas de Kalker son implementadas de la siguiente manera:

$$\begin{bmatrix} F_x \\ F_y \\ M_z \end{bmatrix} = -\begin{bmatrix} Gabc_{11} & 0 & 0 \\ 0 & Gabc_{22} & Gabc_{23}\sqrt{ab} \\ 0 & -Gabc_{23}\sqrt{ab} & Gabc_{33} \end{bmatrix} \begin{bmatrix} \upsilon_x \\ \upsilon_y \\ \Phi \end{bmatrix} \tag{5.8}$$

Donde la fuerza resultante de Kalker, estará dada por la siguiente expresión:

$$F_{Kalker} = F_x\mathbf{t_1^r} + F_y\mathbf{t_2^r} \tag{5.9}$$

$$M_{Kalker} = M_z\mathbf{n^r} \tag{5.10}$$

Las cuales expresarán el vector de momento y de la fuerza de Kalker en el modelo y que se actualizara en el tiempo.

5.4.1. El problema de equilibrio

El problema de equilibrio pasa por resolver las ecuaciones dinámicas y cinemáticas, buscando los valores de las coordenadas y de las incógnitas de enlace,cuando el sistema esta en reposo. En nuestro modelo buscaremos partir del reposo, para así dar una condiciones de inicio que sean compatibles con las restricciones y evitaremos tener posibles aceleraciones o velocidades que lleven a un problema de inestabilidad.

Con lo cual para esto resolvemos las ecuaciones haciendo 0 los valores de las aceleraciones y velocidades generalizadas y la resolveremos para las coordenadas en incógnitas de enlace.

por lo tanto tendremos que nuestro problema a resolver quedará de la siguiente manera:

$$\begin{bmatrix} \boldsymbol{\Phi_q^T\lambda - \delta} \\ \boldsymbol{\gamma} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{5.11}$$

Este sistema de ecuaciones es resuelto con el comando fsolve de MATLAB el cual resuelve sistema de ecuaciones no lineales en MATLAB, partiendo de unos valores iniciales.Esta función buscara resolver hasta encontrar los valores que satisfacen el planteamiento inicial, mas importante nos dará los valores de partida en el que el modelo se encuentra en reposo.

5.4.2. Integración con Kalker

En la sección anterior mostramos las ecuaciones mediante las cuales obtenemos las fuerzas de Kalker, en esta sección dedicaremos como resolveremos estas ecuaciones en nuestro modelo y de que manera hemos implementado el integrador en nuestro sistema.

Para esto partimos de nuestra estructura general del sistema de ecuaciones dinámico el cual tiene la forma:

$$\begin{bmatrix} \boldsymbol{M_{V\dot{q}}} & \boldsymbol{\Phi_q^T} \\ \boldsymbol{\Phi_{\dot{q}}} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\ddot{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\delta} \\ \boldsymbol{\gamma} \end{bmatrix} \tag{5.12}$$

Partimos de la matriz δ que tiene los componentes de las fuerzas que están aplicadas al modelo y que no son de enlace y es donde se encuentran las fuerzas de Kalker.

En este caso tenemos que las fuerzas de Kalker que se introducen al resolver el sistema. Tal como se plantea generan un efecto de stiffness el cual hace que el integrador en el tiempo sea inestable teniendo como consecuencia que paulatinamente nos alejemos de la solución.

Con lo cual la propuesta es reducir este efecto, integrando esta parte de manera implícita, logrando así evitar la inestabilidad numérica.

Para hacer esto, buscamos expresar las fuerzas de Kalker de manera que tomen la forma de una fuerza viscosa:

$$\boldsymbol{F_i^{Kalker}} = -\boldsymbol{C_i^{Kalker}} \boldsymbol{s_i} \tag{5.13}$$

donde

$$\boldsymbol{F_i^{Kalker}} = \begin{bmatrix} F_x \\ F_y \\ M_z \end{bmatrix} \tag{5.14}$$

$$\boldsymbol{s_i} = \begin{bmatrix} (V_{P^w}^{P_w^w} - V_{P^r}^{P_r^r}) t_x^r \\ (V_{P^w}^{P_w^w} - V_{P^r}^{P_r^r}) t_y^r \\ (\omega_{P^w}^{P_w^w} - \omega_{P^r}^{P_r^r}) n^r \end{bmatrix} \tag{5.15}$$

$$\boldsymbol{C_i^{Kalker}} = \frac{Gab}{\frac{1}{2}|V_{P^w}^{P_w^w} + V_{P^r}^{P_r^r}|} \begin{bmatrix} c_{11} & 0 & 0 \\ 0 & c_{22} & ab^{-1/2}c_{23} \\ 0 & -ab^{-1/2}c_{23} & abc_{33} \end{bmatrix} \tag{5.16}$$

Dejamos que las fuerzas de Kalker sean las variables de entradas en el modelo dinámico, estas fuerzas pueden ser expresadas como:

$$\sum_i \frac{\partial \boldsymbol{\delta}}{\partial \boldsymbol{F}_i^{\boldsymbol{Kalker}}} \boldsymbol{C}_i^{\boldsymbol{Kalker}} \frac{\partial \boldsymbol{s_i}}{\partial \dot{\boldsymbol{q}}} \dot{\boldsymbol{q}} = \boldsymbol{C}^{\boldsymbol{Kalker}} \dot{\boldsymbol{q}} \tag{5.17}$$

Con lo cual el excluir la contribución de Kalker en el vector δ, puede ser expresado como:

$$\begin{bmatrix} \boldsymbol{M_q} & \dot{\boldsymbol{\phi}}_{\dot{\boldsymbol{q}}}^{\boldsymbol{T}} \\ \dot{\boldsymbol{\phi}}_{\dot{\boldsymbol{q}}}^{\boldsymbol{T}} & 0 \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\delta} \\ \boldsymbol{\gamma} \end{bmatrix} + \begin{bmatrix} -\boldsymbol{C}^{\boldsymbol{Kalker}} \dot{\boldsymbol{q}} \\ 0 \end{bmatrix} \tag{5.18}$$

De esta manera la contribución lineal de las fuerzas de Kalker para la dinámica son tratadas aparte .Por lo que las ecuaciones quedan expresadas de la siguiente manera:

$$\begin{gathered} \boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{\Phi}_{\boldsymbol{q}}^{\boldsymbol{t}} \boldsymbol{\lambda} = \boldsymbol{\delta_2} - \boldsymbol{C}\dot{\boldsymbol{q}} \\ \boldsymbol{\Phi_q}\ddot{\boldsymbol{q}} = \boldsymbol{\gamma} \end{gathered} \tag{5.19}$$

donde hacemos que:

$$\ddot{\boldsymbol{q}} = \frac{\dot{\boldsymbol{q}}_{(\boldsymbol{t}+\boldsymbol{\Delta t})} - \dot{\boldsymbol{q}}_{\boldsymbol{t}}}{\boldsymbol{\Delta t}} \tag{5.20}$$

sustituyendo en 5.18 tenemos:

$$\boldsymbol{M}\left(\frac{\dot{\boldsymbol{q}}_{(\boldsymbol{t}+\boldsymbol{\Delta t})} - \dot{\boldsymbol{q}}_{\boldsymbol{t}}}{\boldsymbol{\Delta t}}\right) + \boldsymbol{\Phi}_{\boldsymbol{q}}^{\boldsymbol{t}} \boldsymbol{\lambda} = \boldsymbol{\delta_2} - \boldsymbol{C}\dot{\boldsymbol{q}} \tag{5.21}$$

$$\boldsymbol{\Phi_q}\left(\frac{\dot{\boldsymbol{q}}_{(\boldsymbol{t}+\boldsymbol{\Delta t})} - \dot{\boldsymbol{q}}_{\boldsymbol{t}}}{\boldsymbol{\Delta t}}\right) = \boldsymbol{\gamma} \tag{5.22}$$

Despejando y agrupando términos convenientemente nos queda:

$$\begin{bmatrix} \boldsymbol{M} + \boldsymbol{C\Delta t} & \boldsymbol{\Phi}_{\boldsymbol{q}}^{\boldsymbol{T}} \boldsymbol{\Delta t} \\ \boldsymbol{\Phi_q} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_{(\boldsymbol{t}+\boldsymbol{\Delta t})} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\delta \Delta t} + \boldsymbol{M}\dot{\boldsymbol{q}}_{\boldsymbol{t}} \\ \boldsymbol{\gamma \Delta t} + \boldsymbol{\Phi_q}\dot{\boldsymbol{q}}_{\boldsymbol{t}} \end{bmatrix} \tag{5.23}$$

De esta manera resolvemos el problema de velocidades en el instante t+Δt,a partir de esto buscamos la posición en cualquier instante de tiempo con la siguiente ecuación:

$$\boldsymbol{q}_{(\boldsymbol{t}+\boldsymbol{\Delta t})} = \boldsymbol{q}_{\boldsymbol{t}} + \dot{\boldsymbol{q}}_{\boldsymbol{t}} \boldsymbol{\Delta t} + \frac{\mathbf{1}}{\mathbf{2}} \left(\dot{\boldsymbol{q}}_{(\boldsymbol{t}+\boldsymbol{\Delta t})} - \dot{\boldsymbol{q}}_{\boldsymbol{t}}\right) * \boldsymbol{\Delta t} \tag{5.24}$$

5.4.3. Oscilación de Klingel

La oscilación de Klingel no es más que un movimiento usualmente no buscado cerca del equilibrio. Esta expresión describe como un sistema busca el equilibrio.

Un movimiento clásico de este fenómeno es el balanceo de un vehículo ferroviario, causado por la acción de conicidad en dirección de estabilidad debido a

la adhesión ferroviaria. Esto surge de la interacción entre las fuerzas de adhesión y las fuerzas inerciales. A bajas velocidades la adhesión domina, pero a medida que la velocidad incrementa, las fuerzas inerciales y de adhesión pasan a ser comparables en magnitud y la oscilación comienza a una velocidad critica.

Sobre esta velocidad el movimiento puede ser violento dañando las ruedas y causando un posible descarrilamiento.

Si por alguna razón las ruedas se encuentran hacia algún lado, los diámetros de las regiones de contacto y por consiguiente las velocidades tangenciales de las ruedas sean diferentes lo cual tenderá a regresar al centro.

una estimación de la longitud de onda puede ser expresada por la siguiente ecuación:

$$\lambda = 2\pi\sqrt{\frac{2lr_0}{2tan\alpha}} \tag{5.25}$$

donde:

- λ= longitud onda de klingel.
- 2l= distancia entre centros.
- r_0= radio de la rueda en la posición central.
- α=ángulo de conicidad de la rueda.

y la frecuencia de Klingel vendrá dada con la siguiente ecuación:

$$f = V/\lambda \tag{5.26}$$

donde V es la velocidad en la dirección x de las ruedas.

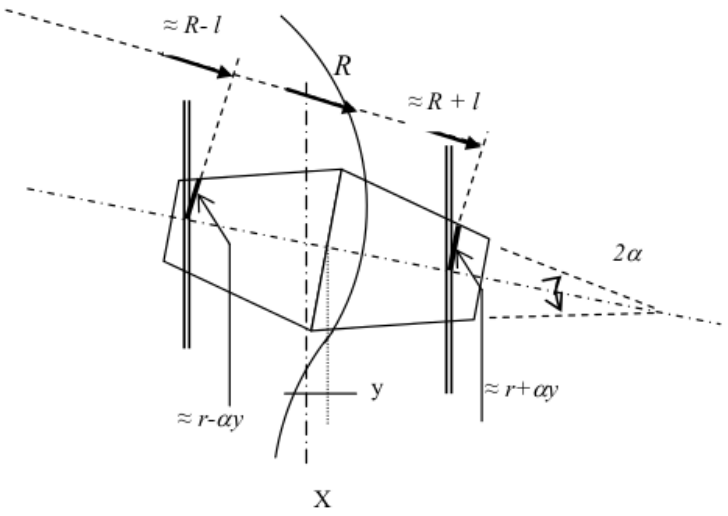

Figura 5.3: Oscilacion de Klingel

Capítulo 6

Validación cualitativa del modelo

6.1. Validación del modelo en condición de operación ideal

Una manera de validar el modelo es dar una condición en la que se tenga un comportamiento esperado, por ello las gráficas que a continuación se analizan muestran datos obtenidos de pruebas bajo distintas condiciones

La evolución de las variables más importantes para determinar si el comportamiento a lo largo del tiempo del modelo es correcto serán las encargadas de demostrar su validez.

6.1.1. Par motor aplicado desde el raíl

Si consideramos desde la posición de equilibrio, dar un par motor a las ruedas del raíl y así ver la transmisión del movimiento a partir de las fuerzas de Kalker, estas teóricamente transmitirán el par torsor de las ruedas del raíl a las ruedas del chasis. El par torsor aplicado es de 2NM

En esta condición de operación representa el comportamiento general del banco de ensayos y es nuestro caso más ideal.

Las coordenadas con las que podremos verificar el comportamiento puede ser la del giro de las ruedas del raíl, el giro de las ruedas del bogie y la evolución en el tiempo de las fuerzas de Kalker nos genera una idea de las cargas. que se transmiten en el banco de pruebas. Por lo que tenemos:

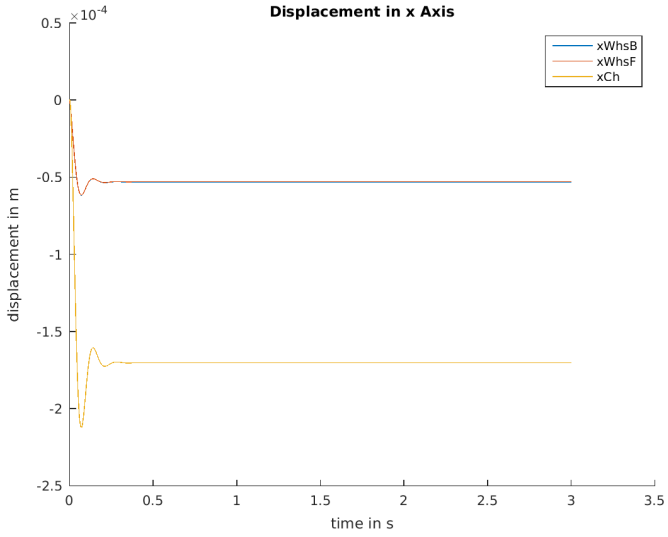

Figura 6.1: Desplazamientos en el eje x del bogie en el tiempo

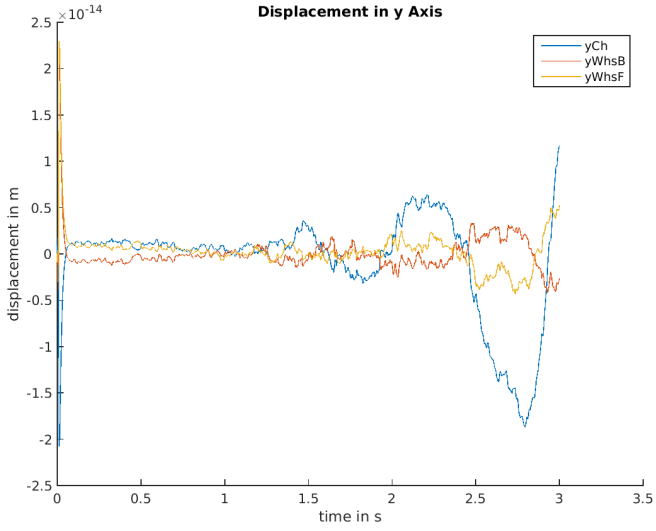

Figura 6.2: Desplazamientos en el eje y del bogie en el tiempo

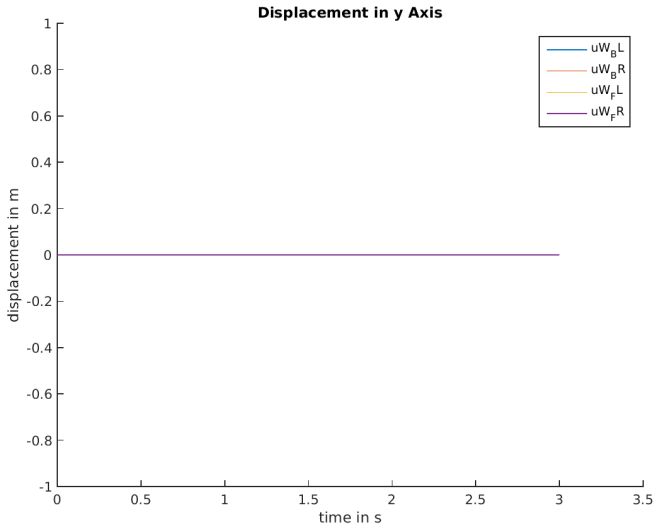

Figura 6.3: Desplazamientos en el eje y de la rueda en el punto de contacto en el tiempo

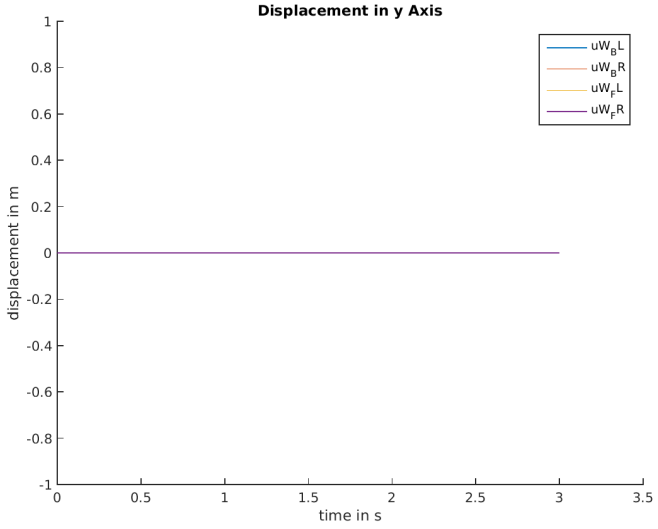

Figura 6.4: Desplazamientos en el eje y del rail en el punto de contacto en el tiempo

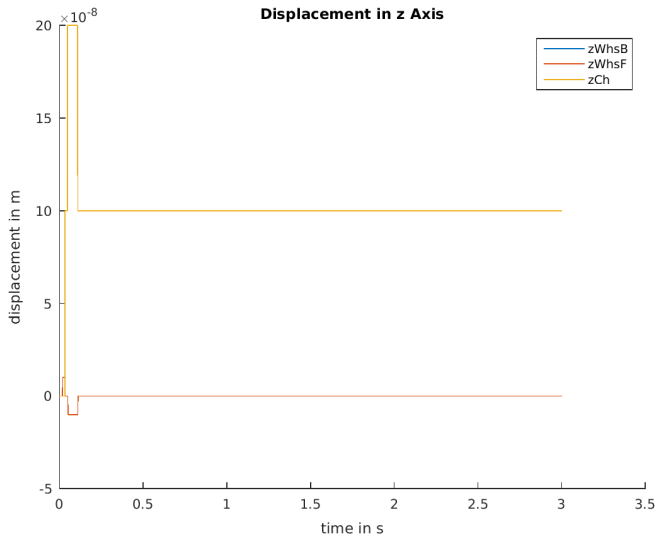

Figura 6.5: Desplazamientos en el eje z del bogie en el tiempo

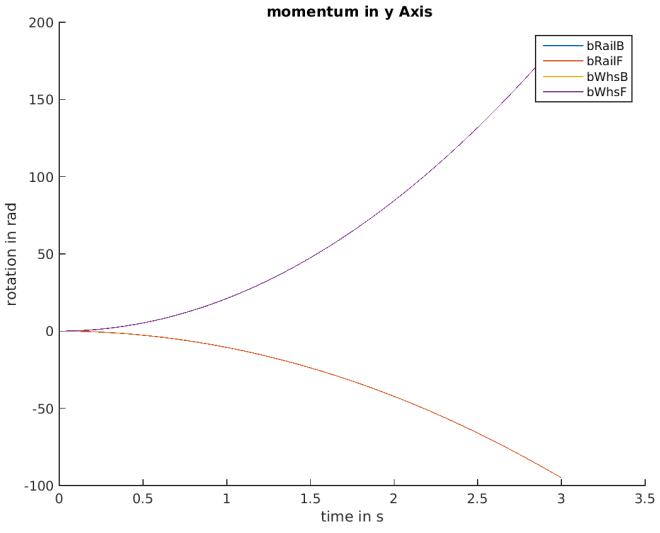

Figura 6.6: Giros en rad de las ruedas y el rail

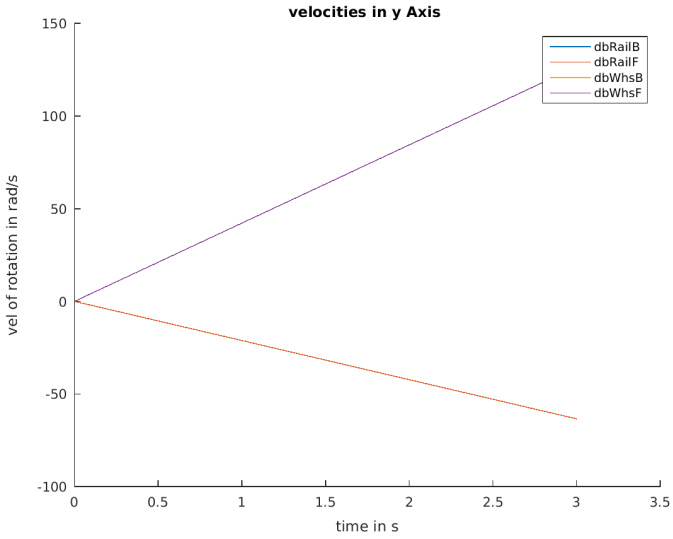

Figura 6.7: Velocidad en rad/s de las ruedas y rail

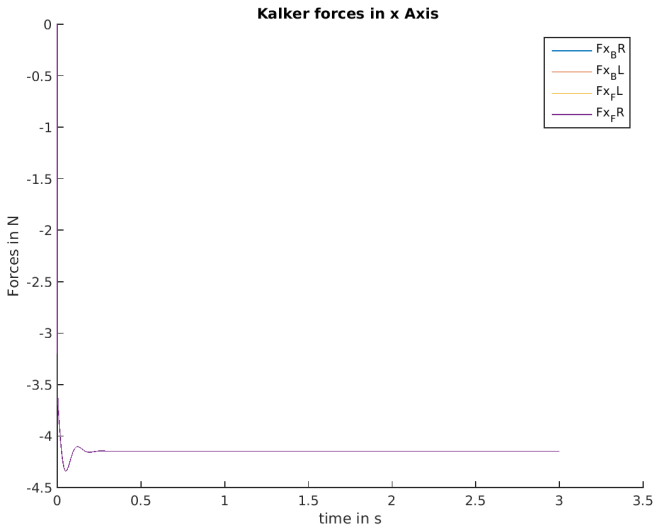

Figura 6.8: Fuerzas de Kalker en la dirección x entre la rueda y el rail en el tiempo

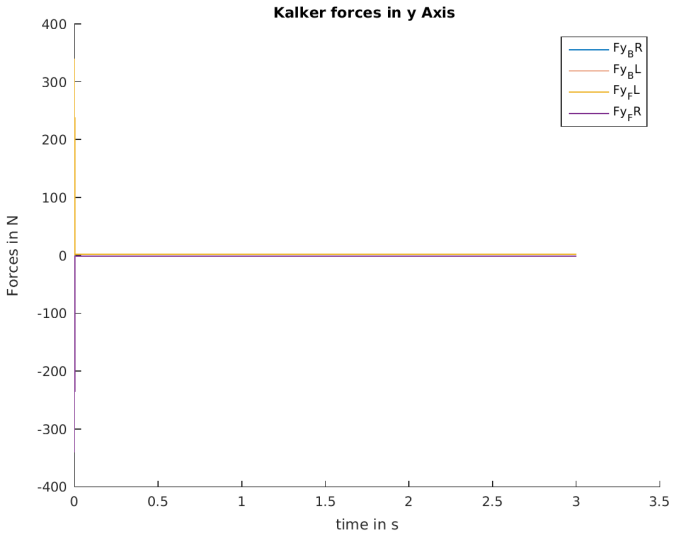

Figura 6.9: Fuerzas de Kalker en la dirección y entre la rueda y el rail en el tiempo

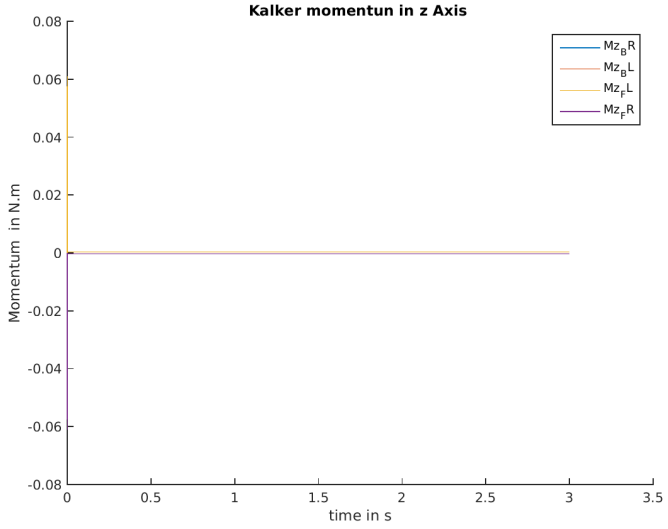

Figura 6.10: Momento de Kalker en la dirección z entre la rueda y el rail en el tiempo

Es importante ver en las fuerzas de Kalker el como partimos de un pico esto es debido a que en un principio el valor del deslizamiento entre la rueda inferior, y la superior es grande con lo cual hace que se produzca una fuerza muy grande la cual es atenuada con un valor limite, este valor limite estará dado en base a la fuerza máxima que se puede dar de una rueda a otra con lo cual nos dará un deslizamiento limite, para esto establecemos el valor limite a la velocidad promedio mínima entre las dos ruedas las cuales nos llevaran a un deslizamiento determinado, evitando así una inestabilidad numérica y limitando así el modelo

6.1.2. Par motor aplicado desde el raíl y un desplazamiento en dirección y del bogie

Esta situación partimos de un desplazamiento inicial en la dirección y de 0.001m del bogie y un par motor al cual mantendremos constante con un par motor en las ruedas de la parte superior que dependerán de la velocidad de giro, buscando tener un balanceo del bogie o un movimiento periódico en el eje y. Con este comportamiento podemos también observar la frecuencia de Klingel, las coordenadas del sistema en función del tiempo quedarán de la siguiente manera:

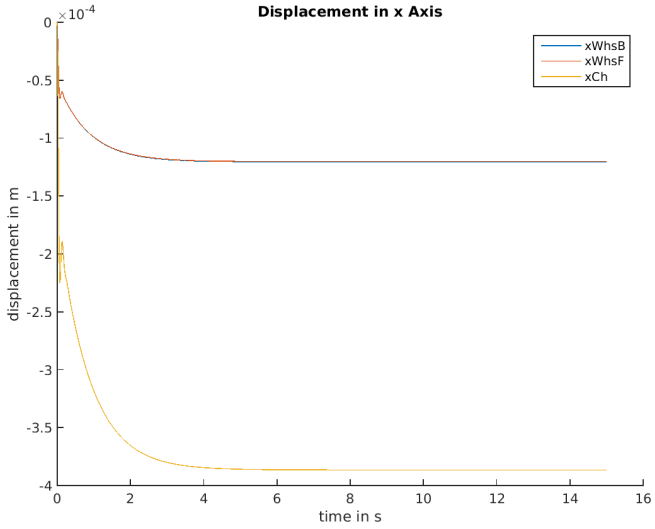

Figura 6.11: Desplazamientos en el eje x del bogie en el tiempo

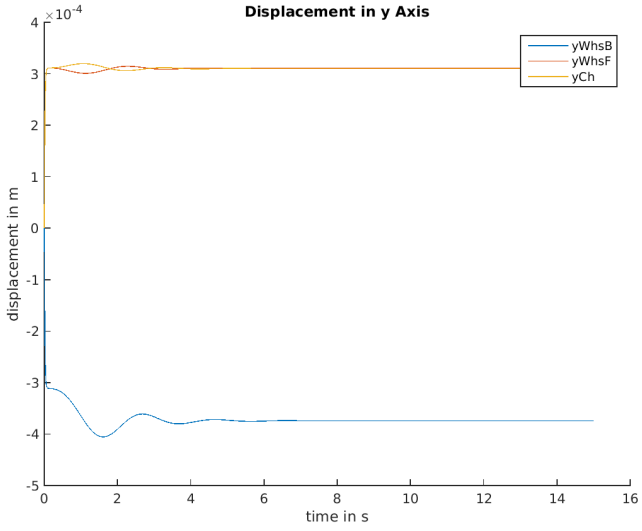

Figura 6.12: Desplazamientos en el eje y del bogie en el tiempo

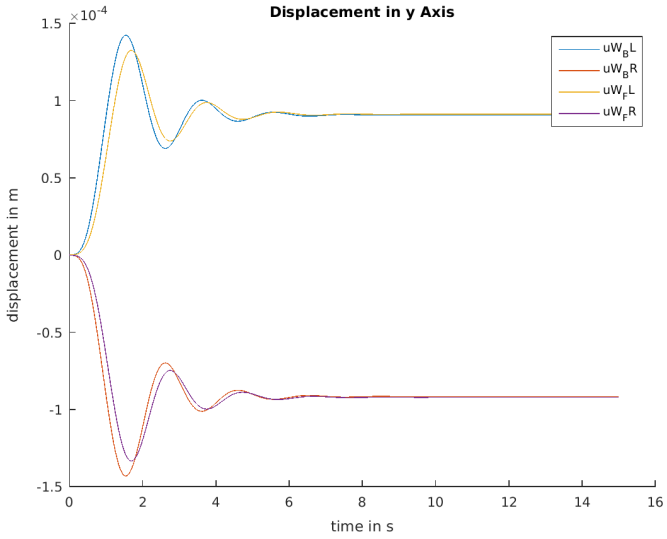

Figura 6.13: Desplazamientos en el eje y de la rueda en el punto de contacto en el tiempo

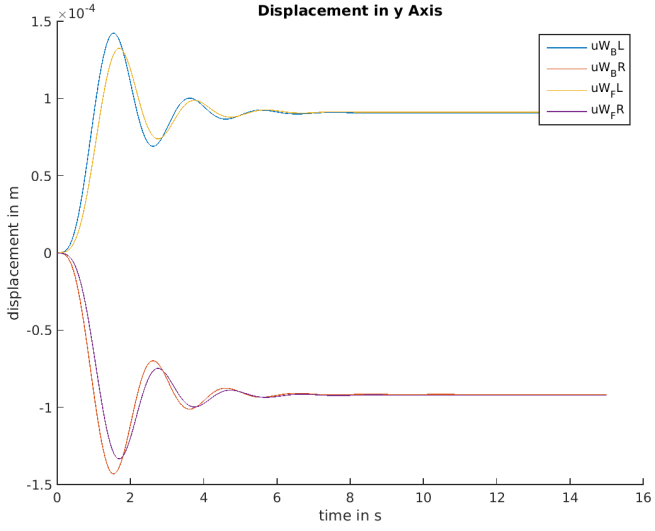

Figura 6.14: Desplazamientos en el eje y del rail en el punto de contacto en el tiempo

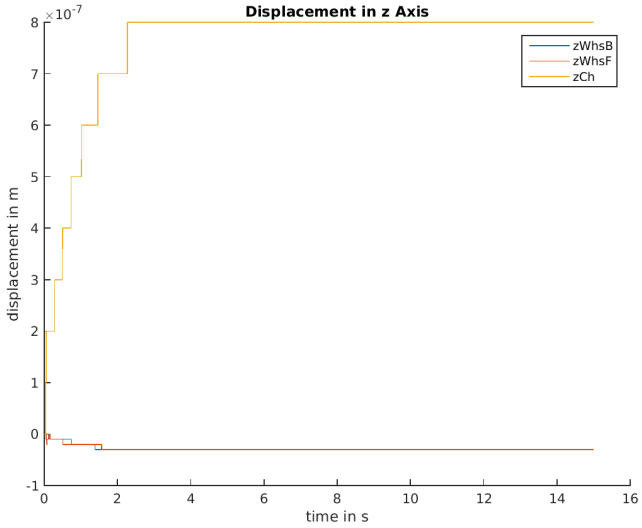

Figura 6.15: Desplazamientos en el eje z del bogie en el tiempo

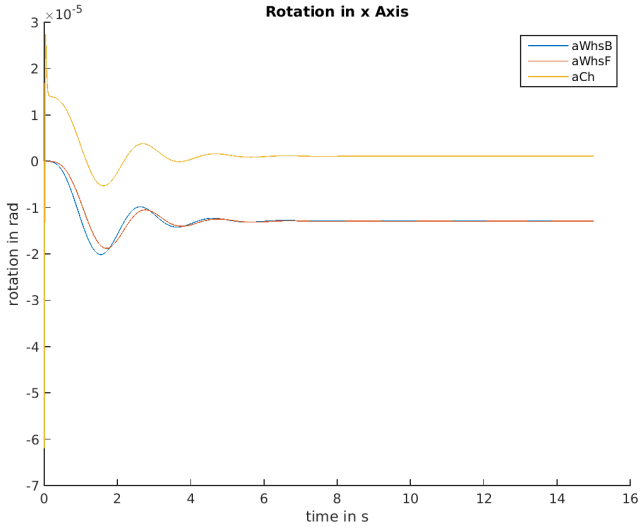

Figura 6.16: Giros en rad alrededor del eje x

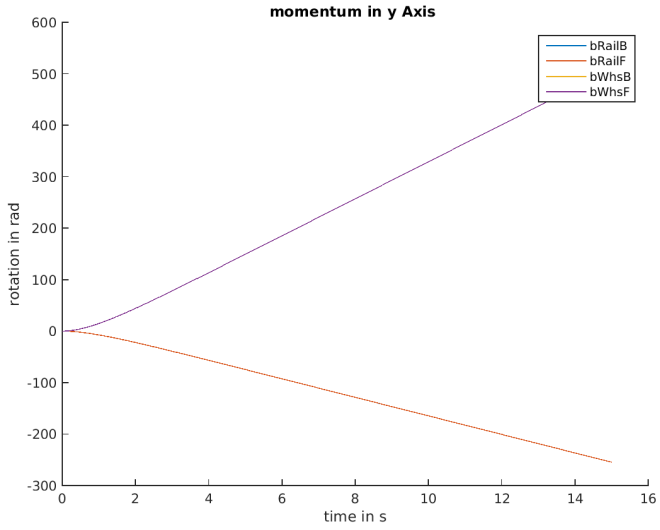

Figura 6.17: Giros en rad de las ruedas y el rail

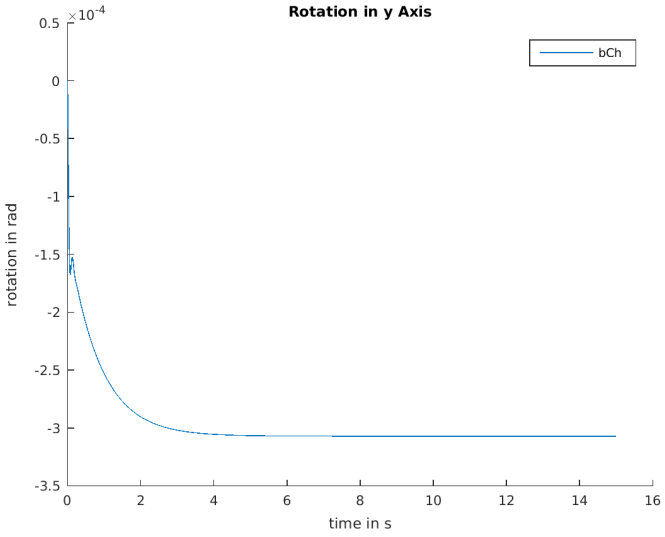

Figura 6.18: Giros en rad alrededor del eje y

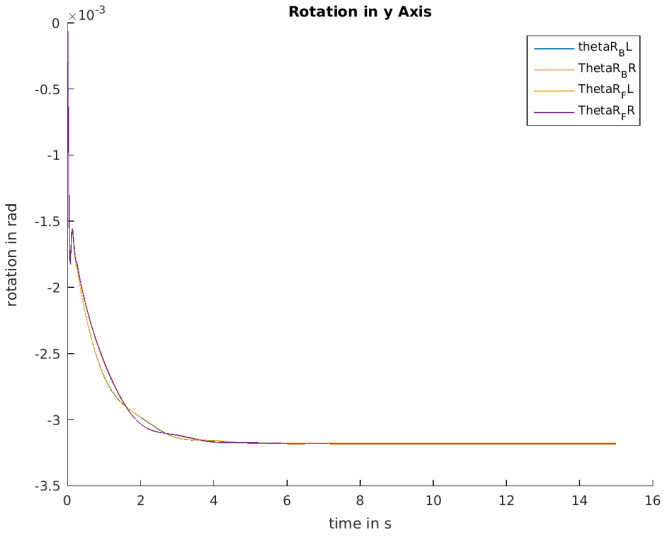

Figura 6.19: Giros en rad alrededor del eje y

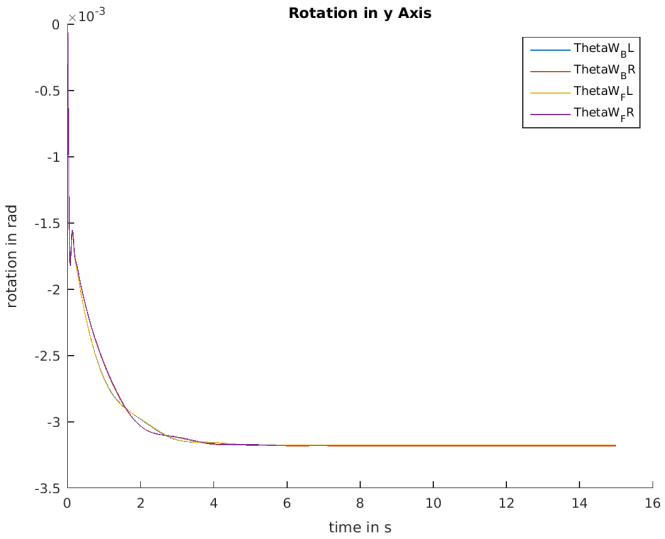

Figura 6.20: Giros en rad alrededor del eje y

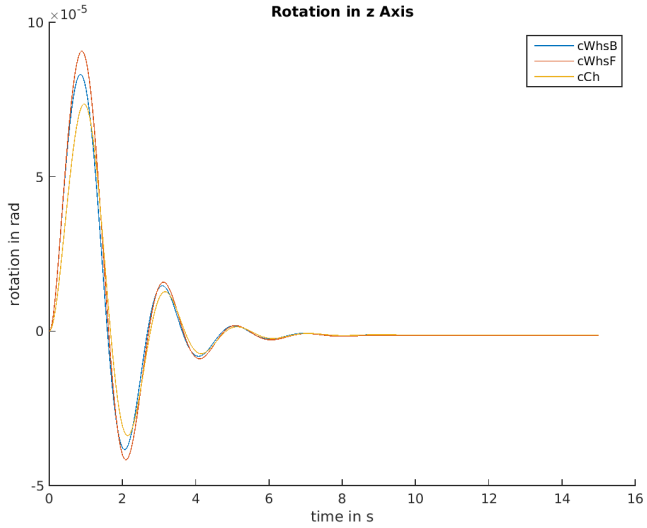

Figura 6.21: Giros en rad alrededor del eje z

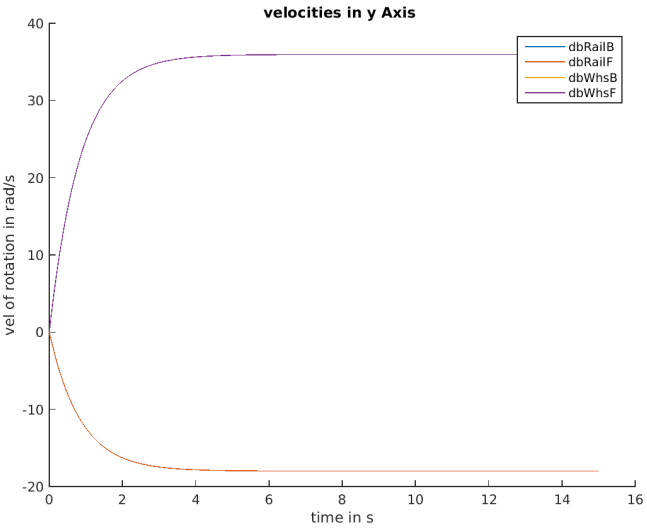

Figura 6.22: Velocidad en rad/s de las ruedas y rail

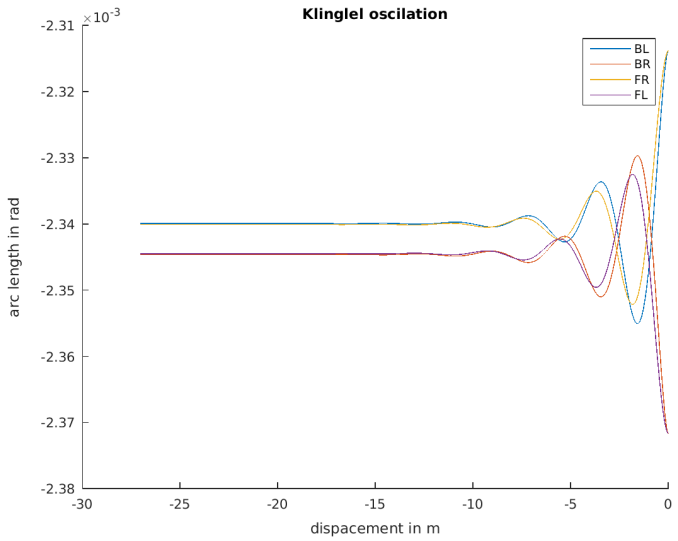

Figura 6.23: longitud de arco vs desplazamiento en U

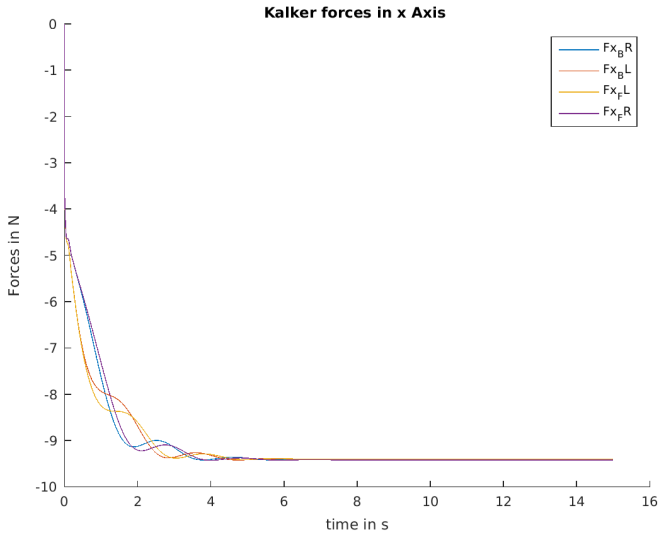

Figura 6.24: Fuerzas de Kalker en la dirección x entre la rueda y el rail en el tiempo

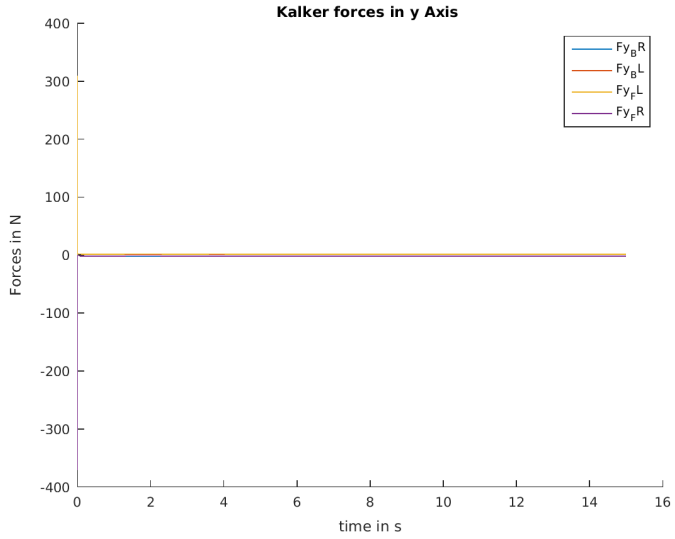

Figura 6.25: Fuerzas de Kalker en la dirección y entre la rueda y el rail en el tiempo

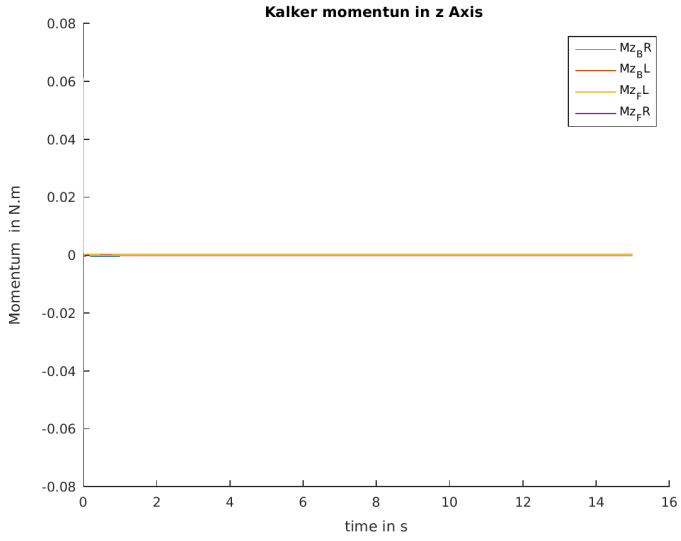

Figura 6.26: Momento de Kalker en la dirección z entre la rueda y el rail en el tiempo

Después de esto una segunda simulación fue llevada a cabo para corroborar el modelo en este caso para el valor de la frecuencia de Klingel la cual a una velocidad distinta se mantuvo en 0,29 hz. Lo cual teóricamente es lo que debe suceder en el modelo.

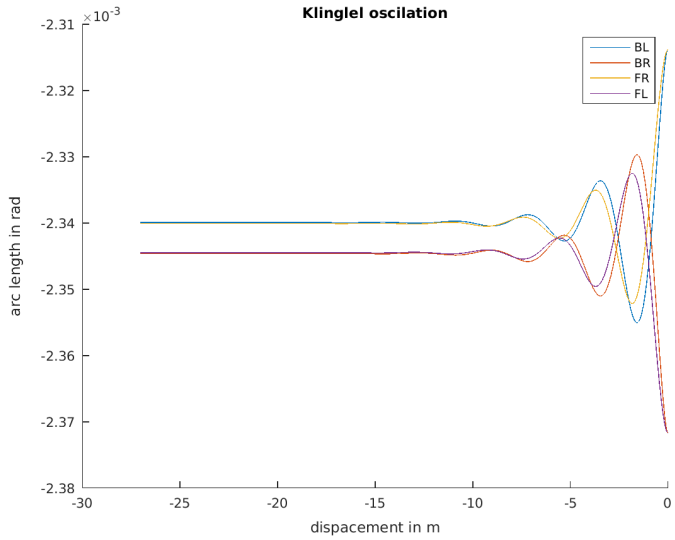

Figura 6.27: longitud de arco vs desplazamiento en U

6.1.3. Inclinación de la bancada

En la siguiente gráficas se muestra el comportamiento de la bancada ante una inclinación de aproximadamente 5 grados respecto del eje x, y moviéndose a una velocidad de 36 rad/s.En donde experimentalmente buscaremos observar algún tipo de desplazamiento con respecto al eje y ya que en un principio se logra ver un movimiento ligero, sin embargo este se atenuara y se mantendrá estable en el tiempo debido a las fuerzas de fricción.

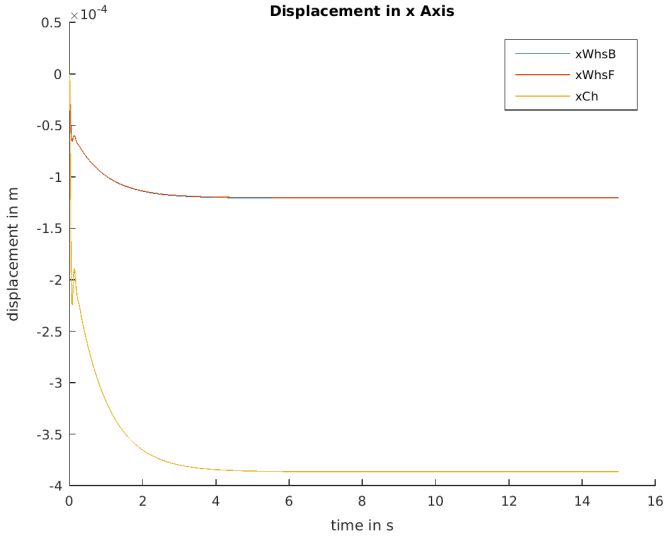

Figura 6.28: Desplazamientos en el eje x del bogie en el tiempo

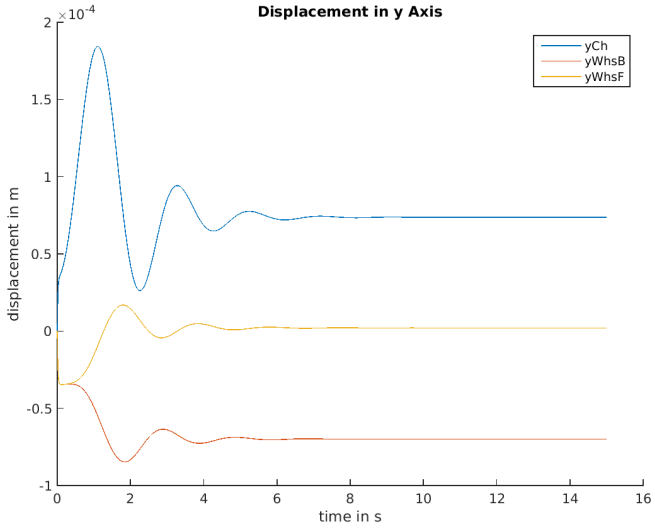

Figura 6.29: Desplazamientos en el eje y del bogie en el tiempo

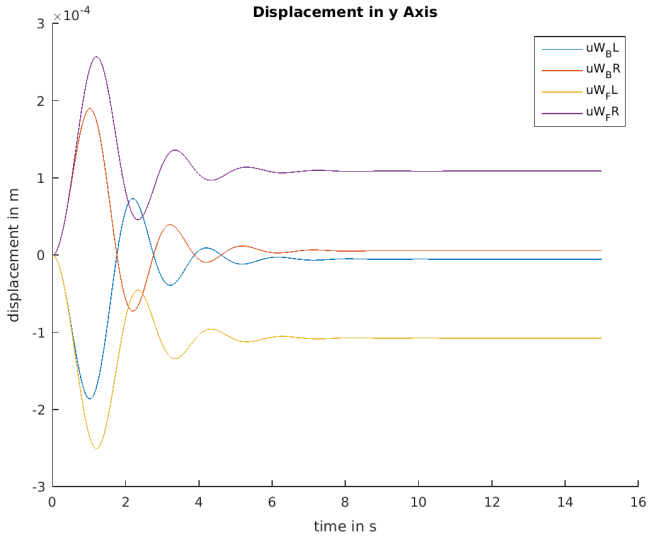

Figura 6.30: Desplazamientos en el eje y de la rueda en el punto de contacto en el tiempo

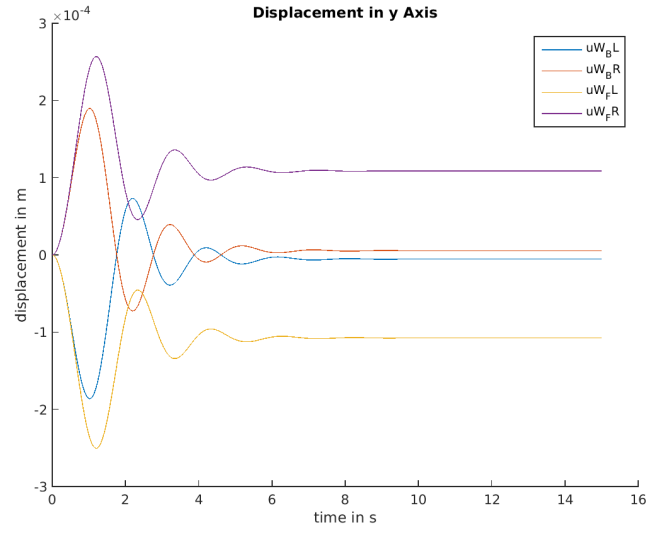

Figura 6.31: Desplazamientos en el eje y del rail en el punto de contacto en el tiempo

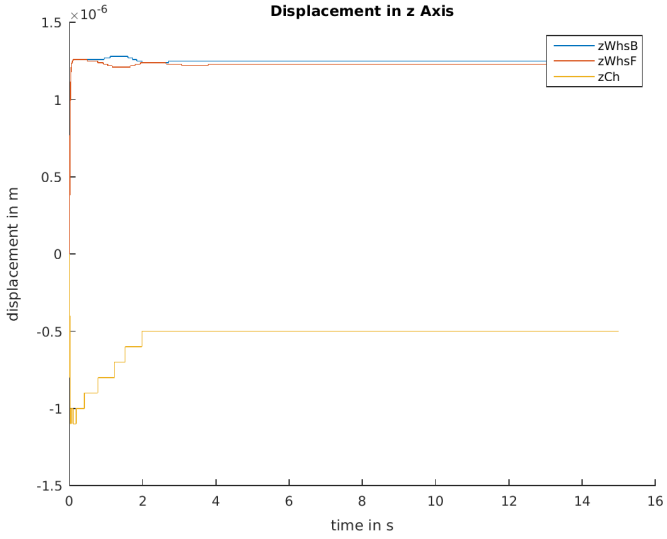

Figura 6.32: Desplazamientos en el eje z del bogie en el tiempo

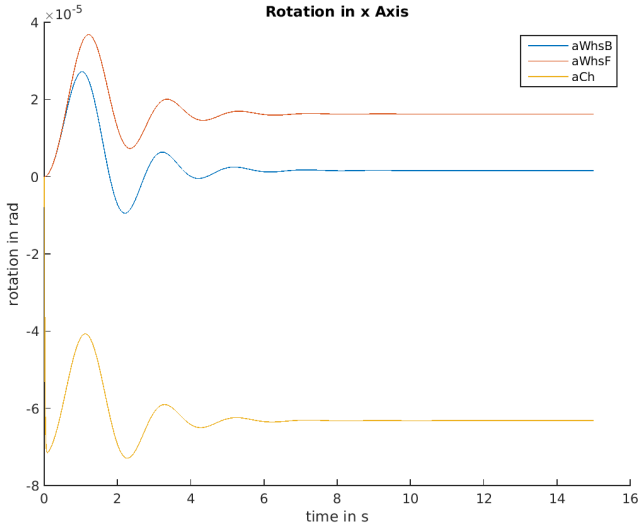

Figura 6.33: Giros en rad alrededor del eje x

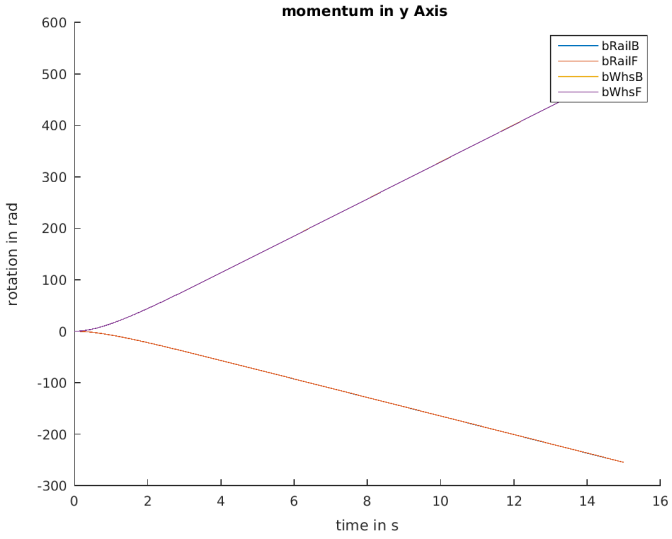

Figura 6.34: Giros en rad de las ruedas y el rail

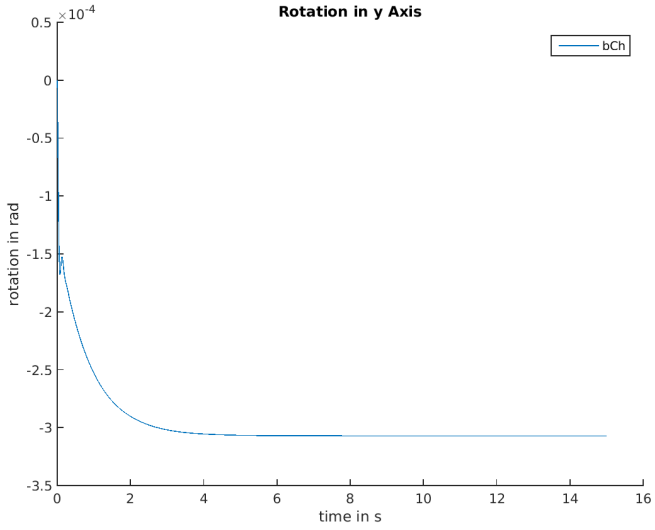

Figura 6.35: Giros en rad alrededor del eje y

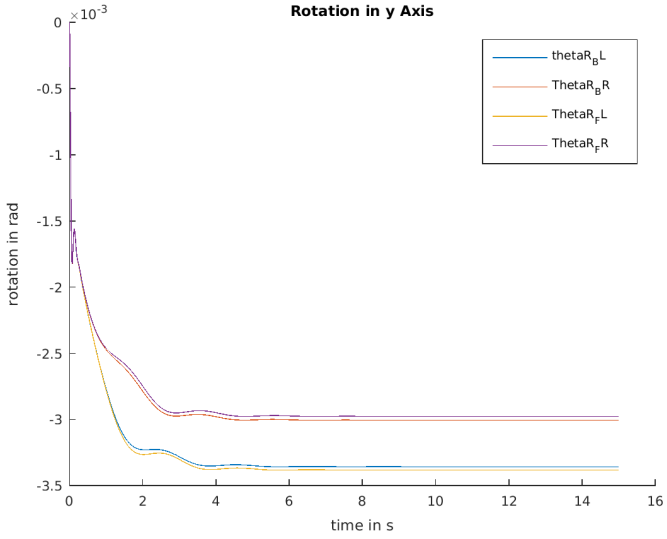

Figura 6.36: Giros en rad alrededor del eje y

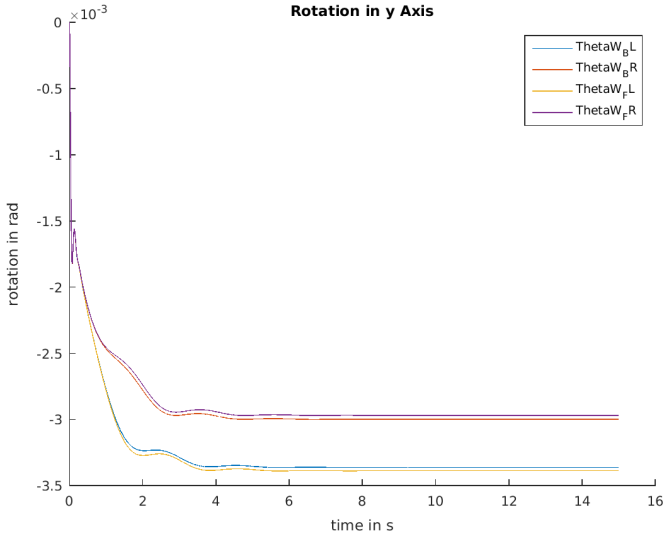

Figura 6.37: Giros en rad alrededor del eje y

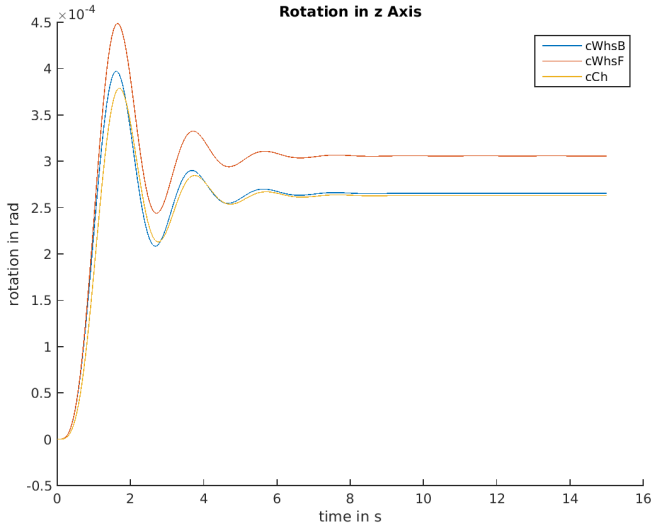

Figura 6.38: Giros en rad alrededor del eje z

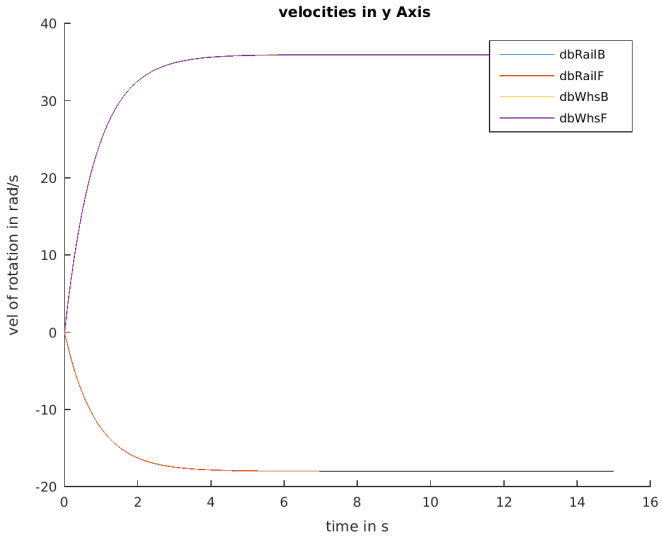

Figura 6.39: Velocidad en rad/s de las ruedas y rail

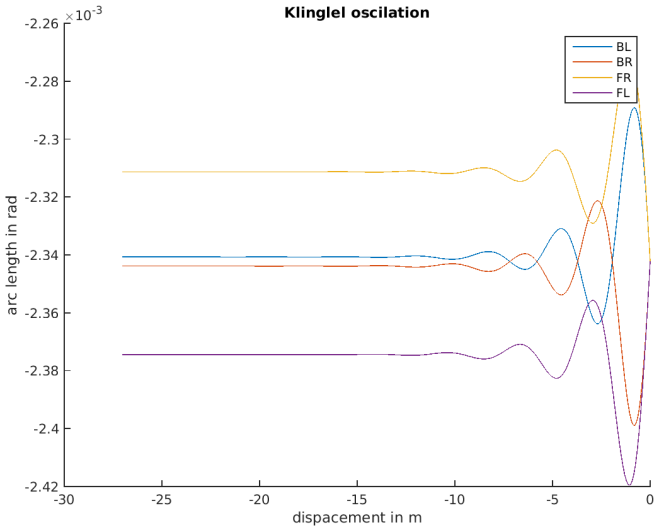

Figura 6.40: longitud de arco vs desplazamiento en U

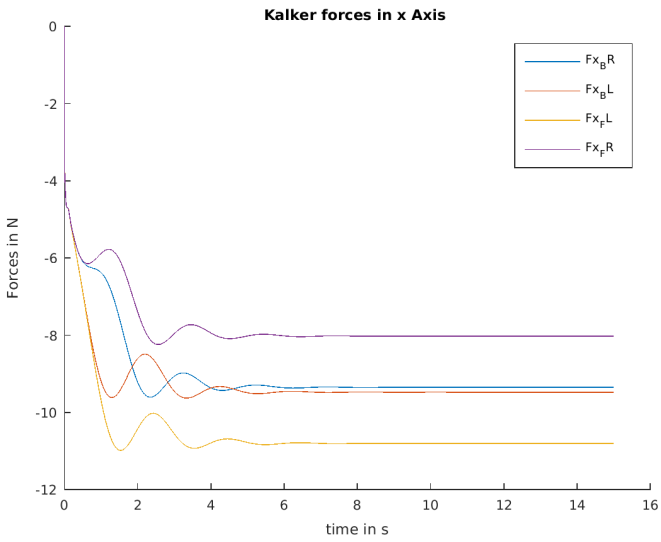

Figura 6.41: Fuerzas de Kalker en la dirección x entre la rueda y el rail en el tiempo

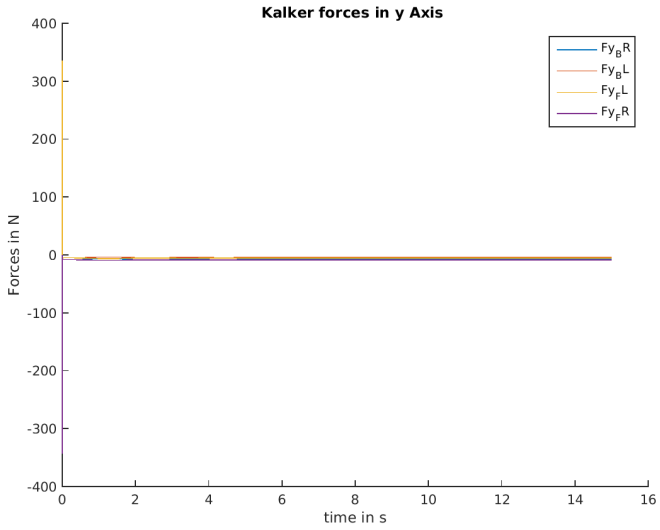

Figura 6.42: Fuerzas de Kalker en la dirección y entre la rueda y el raíl en el tiempo

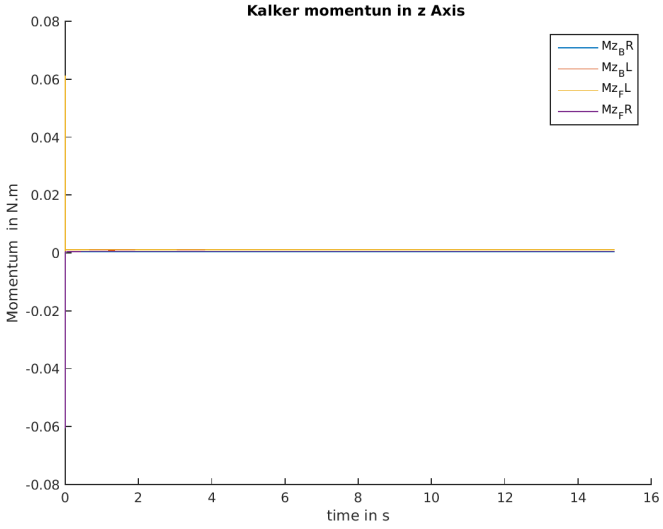

Figura 6.43: Momento de Kalker en la dirección z entre la rueda y el raíl en el tiempo

6.1.4. Conclusiones

En este trabajo se ha desarrollado un modelo de un banco de ensayos de un bogie de tren a escala, el cual fue construido en el laboratorio de ingeniería mecánica aplicada y computacional de la UPNA, con el fin de poder de experimentar y obtener parámetros dinámicos del sistema.

La caracterización fiable de las fuerzas de contacto de fricción que aparecen entre el raíl y la rueda, fue una de las prioridades del proyecto, de forma que el comportamiento del bogie sea lo mas preciso posible.

El modelo de contacto lineal de Kalker es un modelo discreto con el cual podemos obtener las fuerzas, a partir de ciertos parámetros en cada instante de tiempo, entre estos parámetros, tenemos la huella y la fuerza normal ejercida entre el raíl y la rueda.

Es importante ver en cada uno de los casos presentados anteriormente el comportamiento de las fuerzas, ya que en la mayoría de los casos en el estado inicial, podemos ver un pico, esto es debido a que en el instante inicial los pseudo deslizamientos producidos tienden a ser muy altos. en este puntos hemos logrado introducir al modelo un limitador de la fuerza el cual expresa el par máximo que se puede transmitir entre las ruedas.

La simulación dinámica se tuvo que ir al uso de un integrador de Euler explicito-implícito el cual debido a un error de stiffness ayuda a hallar la solución de las fuerzas en el tiempo. El paso de integración (delta t) utilizado para la simulación es de 0.001 s. esto se traduce en que el tiempo de análisis es mayor que el tiempo físico del movimiento, por lo que no es posible realizar simulaciones a tiempo real.

Uno de los casos mas representativos es en el evaluado en el apartado 6.1.1 donde representamos una condición ideal de funcionamiento y logramos ver como se transmiten las fuerzas de fricción de Kalker de una rueda a otra esto se puede ver en la 6.6.Lo cual es muy importante para saber el comportamiento correcto del modelo. En el apartado 6.1.3 responde a un experimento hecho en el laboratorio y logramos ver que el comportamiento de este corresponde con lo que sucede en el modelo.

Con lo cual se pudo constar que la respuesta de nuestra simulación dinámica en los diferentes casos, con respecto a las hipótesis y experimentos realizados son acordes o se corresponden entre estos con lo cual se puede decir que el modelo presentado en el trabajo es valido.

6.1.5. Otras simulaciones y pruebas

Muchas variables y muchas condiciones pueden ser evaluadas , por lo que se ha optado poner estos casos que muestran un comportamiento correcto en situaciones muy ideales

para completar esta validación se consideró entonces colocar un vídeo en el que se muestre diferentes situaciones para el banco de pruebas, durante la simulación no solo podremos ver el comportamiento, también podemos observar los vectores tangentes y normales del punto de contacto así como las fuerzas de

Kalker en su paso por el tiempo, este vídeo formará parte de los anexos que se entreguen del trabajo.

Capítulo 7

Mejoras en el modelo

Aun cuando se tiene un modelo que es capaz de reproducir unas condiciones muy cercanas a las condiciones en las que se estará sometido el banco de pruebas, es posible de alguna manera dar algunas mejoras al modelo, entre estas pueden ser:

7.0.6. Transmisión del Par Motor

una de las condiciones que puede buscarse reproducir es la transmisión del par que ofrece el motor a cada una de las ruedas , las cuales en este modelo están puestas como una entrada al sistema, sin embargo esto puede modificarse de manera que se consideren las perdidas que se tienen por la transmisión debido al sistema de poleas y correas.

7.0.7. Tiempo de integración

Aun cuando se tiene un paso de integración de 0.001, es posible considerar que esto se puede ejecutar de una manera mas rápida, si nos fijamos en las gráficas respecto al tiempo podemos apreciar que para un tiempo menos a 0.5s se tiene una variación muy brusca debido a las fuerzas de Kalker, sin embargo después de este tiempo el modelo se comienza comportar de manera mas estables con lo cual es posible que para tiempos mayores a 0.5 usemos un paso de integración mayor con lo cual pudiésemos obtener una simulación en un tiempo más cercano a el tiempo real.

Bibliografía

[1] Simon Iwnicki *Handbook of Railway Vehicle Dynamics* , Taylor and Francis Group. Estados Unidos, Boca Raton, 2006

[2] Javier Ros *Symbolic Modeling of Multibody Systems For RealtTime Applications* , UPNA. España, Pamplona, 2015

[3] Javier Ros *Apuntes de Dinámica de Sistemas Multicuerpo* , UPNA. España, Pamplona, 2013

[4] David Herreros Sojo *Diseño y Construcción de un Banco de Ensayos para un Bogie de Tren a Escala* , UPNA. España, Pamplona, 2014

Apendíce

Valores de los Parámetros

```
1
2  g = 9.8000000000000007105;
3  aFrame = 0.3491000000000000214;
4  m_RailB = 10.374468999999999497;
5  m_RailF = 10.374468999999999497;
6  m_Ch = 13.610528000000000404;
7  m_WhsB = 9.269349999999999312;
8  m_WhsF = 9.269349999999999312;
9  K_S1xy = 100000.0;
10 K_S1z = 100000.0;
11 C_S1xy = 1713.0;
12 C_S1z = 1713.0;
13 K_S2 = 100000.0;
14 C_S2 = 1713.0;
15 RRAIL = 0.106249999999999997224;
16 RWHS = 0.053124999999999998612;
17 LRAIL = 0.218499999999999999978;
18 LWHS = 0.218499999999999999978;
19 aW_BL = 0.0;
20 bW_BL = 0.0;
21 cW_BL = 0.18199999999999999512;
22 dW_BL = -0.053124999999999998612;
23 lW_BL = 0.0;
24 aR_BL = 0.0;
25 bR_BL = -2.7473000000000000753;
26 cR_BL = 0.0;
27 dR_BL = 0.106249999999999997224;
28 lR_BL = 0.0;
29 aW_BR = 0.0;
30 bW_BR = 0.0;
31 cW_BR = 0.18199999999999999512;
32 dW_BR = -0.053124999999999998612;
33 lW_BR = 0.0;
34 aR_BR = 0.0;
35 bR_BR = -2.7473000000000000753;
36 cR_BR = 0.0;
37 dR_BR = 0.106249999999999997224;
38 lR_BR = 0.0;
39 aW_FL = 0.0;
40 bW_FL = 0.0;
```

```
cW_FL = 0.18199999999999999512;
dW_FL = -0.053124999999999998612;
lW_FL = 0.0;
aR_FL = 0.0;
bR_FL = -2.7473000000000000753;
cR_FL = 0.0;
dR_FL = 0.10624999999999999997224;
lR_FL = 0.0;
aW_FR = 0.0;
bW_FR = 0.0;
cW_FR = 0.18199999999999999512;
dW_FR = -0.053124999999999998612;
lW_FR = 0.0;
aR_FR = 0.0;
bR_FR = -2.7473000000000000753;
cR_FR = 0.0;
dR_FR = 0.10624999999999999997224;
lR_FR = 0.0;
Ct = 10000.0;
E_elastic = 2.1E11;
nu_poisson = 0.27000000000000001776;
G_elastic = 8.07E10;
aBR = 0.0;
bBR = 0.0;
aBL = 0.0;
bBL = 0.0;
aFR = 0.0;
bFR = 0.0;
aFL = 0.0;
bFL = 0.0;
C11BR = 0.0;
C22BR = 0.0;
C23BR = 0.0;
C33BR = 0.0;
C11BL = 0.0;
C22BL = 0.0;
C23BL = 0.0;
C33BL = 0.0;
C11FR = 0.0;
C22FR = 0.0;
C23FR = 0.0;
C33FR = 0.0;
C11FL = 0.0;
C22FL = 0.0;
C23FL = 0.0;
C33FL = 0.0;
```

Coordenadas

```
bRailB = 0.0;
bRailF = 0.0;
xCh = 0.0;
yCh = 0.0;
zCh = 0.0;
```

```
cCh = 0.0;
aCh = 0.0;
bCh = 0.0;
bWhsB = 0.0;
cWhsB = 0.0;
aWhsB = 0.0;
xWhsB = 0.0;
yWhsB = 0.0;
zWhsB = 0.0;
bWhsF = 0.0;
cWhsF = 0.0;
aWhsF = 0.0;
xWhsF = 0.0;
yWhsF = 0.0;
zWhsF = 0.0;
thetaW_BL = 3.1415999999999999481;
uW_BL = 0.0;
uR_BL = 0.0;
thetaR_BL = 0.0;
thetaW_BR = 3.1415999999999999481;
uW_BR = 0.0;
uR_BR = 0.0;
thetaR_BR = 0.0;
thetaW_FL = 3.1415999999999999481;
uW_FL = 0.0;
uR_FL = 0.0;
thetaR_FL = 0.0;
thetaW_FR = 3.1415999999999999481;
uW_FR = 0.0;
uR_FR = 0.0;
thetaR_FR = 0.0;
```

Fichero Símbolico lib_3D_mec

```

#include <fstream>
#include "lib_3d_mec_ginac/lib_3d_mec_ginac.h"
#include <ginac/ginac.h>
#include <sstream>

//
    ********************************************************************************

//  PLEASE AlWAYS DEFINE ORDER & MAPLE & METHOD
//  DEFINED IN "globals.h" + "compilation line or with #define"

//  #define METHOD VIRTUAL_POWER or #define METHOD LAGRANGE  // or
    complilation line -DORDER=CMO or -DORDER=RMO
//  #define ORDER CMO or #define ORDER RMO  // or complilation line
     -DORDER=CMO or -DORDER=RMO
//  #define MAPLE MAPLE_ON or #define MAPLE MAPLE_OFF  // or
    complilation line -DMAPLE=MAPLE_ON -DMAPLE=MAPLE_OFF

//  IF METHOD = 1 ==> VIRTUAL_POWER & IF METHOD = 0 ==> LAGRANGE
```

```
16 //   IF ORDER   = 1 ==> CMO              & IF ORDER   = 0 ==> RMO
17 //   IF MAPLE   = 1 ==> MAPLE_ON         & IF MAPLE   = 0 ==> MAPLE_OFF
18 //
       ********************************************************************************************
19 #ifdef LAG
20   # define METHOD LAGRANGE
21 #endif
22
23 #ifdef V_P
24   # define METHOD VIRTUAL_POWER
25 #endif
26 //
       ********************************************************************************************
27 #ifdef Q
28   #define PROBLEM_TYPE "Q"
29 #endif
30 #ifdef Z
31   #define PROBLEM_TYPE "Z"
32 #endif
33 #ifdef I3AL_Q
34   #define PROBLEM_TYPE "I3AL_Q"
35 #endif
36 #ifdef I3AL_Z
37   #define PROBLEM_TYPE "I3AL_Z"
38 #endif
39 #ifdef ALL
40   #define PROBLEM_TYPE "ALL"
41 #endif
42 //
       ********************************************************************************************
43
44
45 using namespace GiNaC;
46 using namespace std;
47
48 // SYSTEM DEFINITION
49 //
       ********************************************************************************************
50
51 void printError2(const char* arg){
52   printf("%s\n", arg);
53 }
54
55 #ifdef PROCEDURE
56 int C_EXPORT_AS_PROCEDURE=1;
57 #else
58 int C_EXPORT_AS_PROCEDURE=0;
59 #endif
60
61
62 int main(int argc, char *argv[]) {
63
```

```
64 //     ********************************************************************************
65 cout << " " << endl;
66 cout << "*********************************************" << endl;
67 cout << "*              BOGIE PROTOTYPE                *" << endl;
68 cout << "*********************************************" << endl;
69 cout << " " << endl;
70 system("date");
71 cout << " " << endl;
72 //     ********************************************************************************
73
74 if (argc!=3){
75         printf("Error: The program %s takes two parameters: Gravity
       (UP/DOWN) and Atomize (YES/NO)\n",argv[0]);
76         exit(1);
77     }
78
79 if (0==strcmp(argv[1],"DOWN")){
80     gravity=DOWN;
81     cout << "Gravity DOWN" << endl;
82 }
83 else{
84     if (0==strcmp(argv[1],"UP")){
85         gravity=UP;
86         cout << "Gravity UP" << endl;
87     }
88     else
89     {
90         printf("Error: The program %s takes two parameters: Gravity
       (UP/DOWN) and Atomize (YES/NO)\n",argv[0]);
91         exit(1);
92     }
93 }
94
95 if (0==strcmp(argv[2],"YES")){
96         atomization=YES;
97         cout << "Atomize YES" << endl;
98 }
99 else{
100
101     if (0==strcmp(argv[2],"NO")){
102         atomization=NO;
103         cout << "Atomize NO" << endl;
104     }
105     else
106     {
107         printf("Error: The program %s takes two parameters: Gravity
       (UP/DOWN) and Atomize (YES/NO)\n",argv[0]);
108         exit(1);
109     }
110 }
111
112 //     ********************************************************************************
```

```
113
114   double integration_time=strtod(argv[1], NULL);
115   double delta_t= strtod(argv[2], NULL);
116   long int k,steps;
117
118   printf("integration_time %g delta_t %g\n", integration_time,
        delta_t);
119
120 //
      ********************************************************************************
121 //  System definition
122 //
      ********************************************************************************
123
124   System sys(&printError2);
125
126
127 //
      ********************************************************************************
128 //  Coordinate definition
129 //
      ********************************************************************************
130
131
132   //symbol_numeric aFrame = *sys.new_Coordinate("aFrame","daFrame
        ","daFrame", 0.0 , 0.0 , 0.0);
133   symbol_numeric bRailB = *sys.new_Coordinate("bRailB","dbRailB","
        ddbRailB", 0.0 , 0.0 , 0.0);
134   symbol_numeric bRailF = *sys.new_Coordinate("bRailF","dbRailF","
        ddbRailF", 0.0 , 0.0 , 0.0);
135         symbol_numeric xCh = *sys.new_Coordinate("xCh","dxCh","
        ddxCh", 0.0 , 0.0 , 0.0);
136         symbol_numeric yCh = *sys.new_Coordinate("yCh","dyCh","
        ddyCh", 0.0 , 0.0 , 0.0);
137         symbol_numeric zCh = *sys.new_Coordinate("zCh","dzCh","
        ddzCh", 0.0 , 0.0 , 0.0);
138         symbol_numeric cCh = *sys.new_Coordinate("cCh","dcCh","
        ddcCh", 0.0 , 0.0 , 0.0);
139         symbol_numeric aCh = *sys.new_Coordinate("aCh","daCh","
        ddaCh", 0.0 , 0.0 , 0.0);
140         symbol_numeric bCh = *sys.new_Coordinate("bCh","dbCh","
        ddbCh", 0.0 , 0.0 , 0.0);
141         symbol_numeric bWhsB = *sys.new_Coordinate("bWhsB","dbWhsB"
        ,"ddbWhsB", 0.0 , 0.0 , 0.0);
142
143
144   symbol_numeric cWhsB = *sys.new_Coordinate("cWhsB","dcWhsB","
        ddcWhsB", 0.0 , 0.0 , 0.0);
145         symbol_numeric aWhsB = *sys.new_Coordinate("aWhsB","daWhsB"
        ,"ddaWhsB", 0.0 , 0.0 , 0.0);
146   symbol_numeric xWhsB = *sys.new_Coordinate("xWhsB","dxWhsB","
        ddxWhsB", 0.0 , 0.0 , 0.0);
```

```
147         symbol_numeric yWhsB = *sys.new_Coordinate("yWhsB","dyWhsB"
      ,"ddyWhsB", 0.0 , 0.0 , 0.0);
148         symbol_numeric zWhsB = *sys.new_Coordinate("zWhsB","dzWhsB"
      ,"ddzWhsB", 0.0 , 0.0 , 0.0);
149
150         symbol_numeric bWhsF = *sys.new_Coordinate("bWhsF","dbWhsF"
      ,"ddbWhsF", 0.0 , 0.0 , 0.0);
151   symbol_numeric cWhsF = *sys.new_Coordinate("cWhsF","dcWhsF","
      ddcWhsF", 0.0 , 0.0 , 0.0);
152         symbol_numeric aWhsF= *sys.new_Coordinate("aWhsF","daWhsF",
      "ddaWhsF", 0.0 , 0.0 , 0.0);
153   symbol_numeric xWhsF = *sys.new_Coordinate("xWhsF","dxWhsF","
      ddxWhsF", 0.0 , 0.0 , 0.0);
154         symbol_numeric yWhsF = *sys.new_Coordinate("yWhsF","dyWhsF"
      ,"ddyWhsF", 0.0 , 0.0 , 0.0);
155         symbol_numeric zWhsF = *sys.new_Coordinate("zWhsF","dzWhsF"
      ,"ddzWhsF", 0.0 , 0.0 , 0.0);
156
157
158   symbol_numeric *dbRailB=sys.get_Velocity("dbRailB");
159         symbol_numeric *dbRailF=sys.get_Velocity("dbRailF");
160         symbol_numeric *dxCh=sys.get_Velocity("dxCh");
161         symbol_numeric *dyCh=sys.get_Velocity("dyCh");
162         symbol_numeric *dzCh=sys.get_Velocity("dzCh");
163         symbol_numeric *dcCh=sys.get_Velocity("dcCh");
164         symbol_numeric *daCh=sys.get_Velocity("daCh");
165         symbol_numeric *dbCh=sys.get_Velocity("dbCh");
166
167         symbol_numeric *dbWhsF=sys.get_Velocity("dbWhsF");
168   symbol_numeric *dcWhsF=sys.get_Velocity("dcWhsF");
169   symbol_numeric *daWhsF=sys.get_Velocity("daWhsF");
170   symbol_numeric *dxWhsF=sys.get_Velocity("dxWhsF");
171         symbol_numeric *dyWhsF=sys.get_Velocity("dyWhsF");
172         symbol_numeric *dzWhsF=sys.get_Velocity("dzWhsF");
173
174         symbol_numeric *dbWhsB=sys.get_Velocity("dbWhsB");
175   symbol_numeric *dcWhsB=sys.get_Velocity("dcWhsB");
176   symbol_numeric *daWhsB=sys.get_Velocity("daWhsB");
177   symbol_numeric *dxWhsB=sys.get_Velocity("dxWhsB");
178         symbol_numeric *dyWhsB=sys.get_Velocity("dyWhsB");
179         symbol_numeric *dzWhsB=sys.get_Velocity("dzWhsB");
180
181
182
183
184
185
186
187   symbol_numeric *ddbRailB=sys.get_Acceleration("ddbRailB");
188         symbol_numeric *ddbRailF=sys.get_Acceleration("ddbRailF");
189         symbol_numeric *ddxCh=sys.get_Acceleration("ddxCh");
190         symbol_numeric *ddyCh=sys.get_Acceleration("ddyCh");
191         symbol_numeric *ddzCh=sys.get_Acceleration("ddzCh");
192         symbol_numeric *ddcCh=sys.get_Acceleration("ddcCh");
193         symbol_numeric *ddaCh=sys.get_Acceleration("ddaCh");
194         symbol_numeric *ddbCh=sys.get_Acceleration("ddbCh");
195
```

```
196         symbol_numeric *ddbWhsF=sys.get_Acceleration("ddbWhsF");
197   symbol_numeric *ddcWhsF=sys.get_Acceleration("ddcWhsF");
198   symbol_numeric *ddaWhsF=sys.get_Acceleration("ddaWhsF");
199   symbol_numeric *ddxWhsF=sys.get_Acceleration("ddxWhsF");
200         symbol_numeric *ddyWhsF=sys.get_Acceleration("ddyWhsF");
201         symbol_numeric *ddzWhsF=sys.get_Acceleration("ddzWhsF");
202
203     symbol_numeric *ddbWhsB=sys.get_Acceleration("ddbWhsB");
204   symbol_numeric *ddcWhsB=sys.get_Acceleration("ddcWhsB");
205   symbol_numeric *ddaWhsB=sys.get_Acceleration("ddaWhsB");
206   symbol_numeric *ddxWhsB=sys.get_Acceleration("ddxWhsB");
207         symbol_numeric *ddyWhsB=sys.get_Acceleration("ddyWhsB");
208         symbol_numeric *ddzWhsB=sys.get_Acceleration("ddzWhsB");
209
210
211
212
213
214   lst coord_indep_init;
215         //coord_indep_init=bRailB-0.0,bRailF-0.0,yCh+0.0,cCh-0.0,
      aCh+0.0,bCh-0.0,cWhsB-0.0,xWhsB-(-0.15625),cWhsF-0.0,xWhsF
      -(0.15625),zCh-0.23184,,yWhsB-0.0,yWhsF-0.0;
216         coord_indep_init=zCh-0.2366;// yCh-0.01,cCh-0.1,bRailB-0.0,
      bRailF-0.0,xCh-0.0,yCh-0.0,zCh-0.2366,cCh-0.0,aCh-0.0,bCh-0.0,
      bWhsB-0.0,cWhsB-0.0,aWhsB-0.0,xWhsB-0.1567,yWhsB-0.0,zWhsB
      +0.0772,bWhsF-0.0,cWhsF-0.0,aWhsF-0.0,xWhsF+0.1567,yWhsF-0.0,
      zWhsF+0.0772;//zCh-0.2366; //xWhsF-(-0.15625),xWhsB-(0.15625);
217   lst vel_indep_init;
218   vel_indep_init= *dbRailB-0.0,*dbRailF-0.0, *dbWhsF-0.0,*dbWhsB
      -0.0;
219
220
221
222 //
      ************************************************************************************
223 //  Kinematical parameter definition
224 //
      ************************************************************************************
225
226   //symbol_numeric d = *sys.new_Parameter("d",0.5);
227   //symbol_numeric h = *sys.new_Parameter("h",0.5);
228
229  symbol_numeric  aFrame = *sys.new_Parameter("aFrame",0.3491);
230
231 //
      ************************************************************************************
232 //   KINEMATICS
233 //
      ************************************************************************************
234
235 //
      ************************************************************************************
```

```
236 //  Define Bases
237 //
        ********************************************************************************************

238
239   sys.new_Base("xyz1","xyz",1,0,0,aFrame);
240   sys.new_Base("B_RailB","xyz1",0,1,0,bRailB);
241   sys.new_Base("B_RailF","xyz1",0,1,0,bRailF);
242   sys.new_Base("B_Ch_c","xyz1",0,0,1,cCh);
243   sys.new_Base("B_Ch_a","B_Ch_c",1,0,0,aCh);
244   sys.new_Base("B_Ch","B_Ch_a",0,1,0,bCh);
245
246   sys.new_Base("B_WhsB_c","xyz1",0,0,1,cWhsB);
247   sys.new_Base("B_WhsB_a","B_WhsB_c",1,0,0,aWhsB);
248   sys.new_Base("B_WhsB","B_WhsB_a",0,1,0,bWhsB);
249
250   sys.new_Base("B_WhsF_c","xyz1",0,0,1,cWhsF);
251   sys.new_Base("B_WhsF_a","B_WhsF_c",1,0,0,aWhsF);
252   sys.new_Base("B_WhsF","B_WhsF_a",0,1,0,bWhsF);
253
254 //
        ********************************************************************************************

255 //  Define Vectors
256 //
        ********************************************************************************************

257
258
259   Vector3D P_O_O_RailB = *sys.new_Vector3D("P_O_O_RailB"
        ,-0.15625,0,0,"xyz1");
260   Vector3D P_O_O_RailF = *sys.new_Vector3D("P_O_O_RailF"
        ,0.15625,0,0,"xyz1");
261   //Vector3D O_OCh = *sys.new_Vector3D("O_OCh",0,0,0.23318,"xyz1");
262         Vector3D P_O_O_Ch = *sys.new_Vector3D("P_O_O_Ch",xCh,yCh,
        zCh,"xyz1");
263
264
265
266   Vector3D P_O_Ch_O_WhsB = *sys.new_Vector3D("P_O_Ch_O_WhsB",xWhsB,
        yWhsB,zWhsB,"xyz1");//*sys.new_Vector3D("OCh_OWHSR
        ",0.15625,0,-0.0738,"B_Ch");
267   //Vector3D P_O_WhsB_O_WhsB = *sys.new_Vector3D("P_O_WhsB_O_WhsB
        ",0,0,0,"xyz1");
268
269   Vector3D P_O_Ch_O_WhsF = *sys.new_Vector3D("P_O_Ch_O_WhsF",xWhsF,
        yWhsF,zWhsF,"xyz1");//*sys.new_Vector3D("OCh_O_WhsF
        ",-0.15625,0,-0.0738,"B_Ch")
270   //Vector3D P_O_WhsF_O_WhsF = *sys.new_Vector3D("P_O_WhsF_O_WhsF
        ",0,0,0,"xyz1");
271
272
273   Vector3D P_O_WhsB_C_BearBL_WhsB = *sys.new_Vector3D("
        P_O_WhsB_C_BearBL_WhsB",0,0.188,0,"B_WhsB_a");
274   Vector3D P_O_WhsB_C_BearBR_WhsB = *sys.new_Vector3D("
        P_O_WhsB_C_BearBR_WhsB",0,-0.188,0,"B_WhsB_a");
```

```
275    Vector3D P_O_WhsF_C_BearFL_WhsF = *sys.new_Vector3D("
         P_O_WhsF_C_BearFL_WhsF",0,0.188,0,"B_WhsF_a");
276    Vector3D P_O_WhsF_C_BearFR_WhsF = *sys.new_Vector3D("
         P_O_WhsF_C_BearFR_WhsF",0,-0.188,0,"B_WhsF_a");
277
278
279
280    //Vector3D X_Ch_STRUCTURE = *sys.new_Vector3D("X_Ch_STRUCTURE
         ",0,0,0,"xyz1");
281    //Vector3D AUX_STRUCTURE = *sys.new_Vector3D("AUX_STRUCTURE
         ",1,0,0,"B_Ch");
282
283    //Rear wheel vectors:
284    Vector3D P_O_Ch_C_BearBL_Ch = *sys.new_Vector3D("
         P_O_Ch_C_BearBL_Ch",-0.15625,0.188,-0.0775,"B_Ch");
285    Vector3D P_O_Ch_C_BearBR_Ch = *sys.new_Vector3D("
         P_O_Ch_C_BearBR_Ch",-0.15625,-0.188,-0.0775,"B_Ch");
286
287    //Front wheel vectors:
288    Vector3D P_O_Ch_C_BearFL_Ch = *sys.new_Vector3D("
         P_O_Ch_C_BearFL_Ch",0.15625,0.188,-0.0775,"B_Ch");
289    Vector3D P_O_Ch_C_BearFR_Ch = *sys.new_Vector3D("
         P_O_Ch_C_BearFR_Ch",0.15625,-0.188,-0.0775,"B_Ch");
290
291
292
293
294
295
296    /*
297
298    Vector3D OCh_K_RF1 = *sys.new_Vector3D("OCh_K_RF1",-0.15625,0,0,"
         B_Ch");
299    Vector3D OCh_K_RR1 = *sys.new_Vector3D("OCh_K_RR1",0.15625,0,0,"
         B_Ch");
300
301    Vector3D OCh_K_RF2 = *sys.new_Vector3D("OCh_K_RF2
         ",-0.15625-0.0477,0.188,0,"B_Ch");
302
303    Vector3D OCh_K_LF1 = *sys.new_Vector3D("OCh_K_LF1
         ",-0.15625+0.0477,-0.188,0,"B_Ch");
304    Vector3D OCh_K_LF2 = *sys.new_Vector3D("OCh_K_LF2
         ",-0.15625-0.0477,-0.188,0,"B_Ch");
305
306  /*
307    Vector3D OCh_OBEAR_LR = *sys.new_Vector3D("OCh_OBEAR_LR
         ",0.15625,0.00000,-0.0738,"B_Ch");-0.0738
308    Vector3D OCh_OBEAR_RR = *sys.new_Vector3D("OCh_OBEAR_RR
         ",0.15625,0.188,-0.0738,"B_Ch");
309    Vector3D OCh_OBEAR_LF = *sys.new_Vector3D("OCh_OBEAR_LF
         ",-0.15625,-0.188,-0.0738,"B_Ch");
310    Vector3D OCh_OBEAR_RF = *sys.new_Vector3D("OCh_OBEAR_RF
         ",-0.15625,0.188,-0.0738,"B_Ch");
311  */
312
313  //
       ********************************************************************************
```

```
314 //  Define Points
315 //
    ********************************************************************************

316
317   Point * O_RailB = sys.new_Point("O_RailB","O",&P_O_O_RailB);
318         Point * O_RailF = sys.new_Point("O_RailF","O",&P_O_O_RailF)
      ;
319         Point * O_Ch = sys.new_Point("O_Ch","O",&P_O_O_Ch);
320         Point * O_WhsB = sys.new_Point("O_WhsB","O_Ch",&
      P_O_Ch_O_WhsB);
321   Point * O_WhsF = sys.new_Point("O_WhsF","O_Ch",&P_O_Ch_O_WhsF);
322
323
324
325         Point * C_BearBL_WhsB = sys.new_Point("C_BearBL_WhsB","
      O_WhsB",&P_O_WhsB_C_BearBL_WhsB);
326   Point * C_BearBR_WhsB = sys.new_Point("C_BearBR_WhsB","O_WhsB",&
      P_O_WhsB_C_BearBR_WhsB);
327         Point * C_BearFL_WhsF = sys.new_Point("C_BearFL_WhsF","
      O_WhsF",&P_O_WhsF_C_BearFL_WhsF);
328         Point * C_BearFR_WhsF = sys.new_Point("C_BearFR_WhsF","
      O_WhsF",&P_O_WhsF_C_BearFR_WhsF);
329
330
331   Point * C_BearBL_Ch = sys.new_Point("C_BearBL_Ch","O_Ch",&
      P_O_Ch_C_BearBL_Ch);
332   Point * C_BearBR_Ch = sys.new_Point("C_BearBR_Ch","O_Ch",&
      P_O_Ch_C_BearBR_Ch);
333   Point * C_BearFL_Ch = sys.new_Point("C_BearFL_Ch","O_Ch",&
      P_O_Ch_C_BearFL_Ch);
334   Point * C_BearFR_Ch = sys.new_Point("C_BearFR_Ch","O_Ch",&
      P_O_Ch_C_BearFR_Ch);
335
336
337
338 //
    ********************************************************************************

339 //  Dynamical Parameter Definition
340 //
    ********************************************************************************

341         //RailB
342         symbol_numeric  m_RailB = *sys.new_Parameter("m_RailB"
      ,10.374469);
343   Vector3D P_O_RailB_G_RailB= *sys.new_Vector3D("P_O_RailB_G_RailB"
      ,0.0,0.0,0.0,"B_RailB");
344       Tensor3D I_RailB = *sys.new_Tensor3D("I_RailB",(ex)0.177926,(
      ex)0.0,(ex)0.0,(ex)0.0,(ex)0.05294,(ex)0.0,(ex)0.0,(ex)0.0,(ex)
      0.177927,"B_RailB"); //Desde origen
345
346         //RailF
347         symbol_numeric  m_RailF = *sys.new_Parameter("m_RailF"
      ,10.374469);
```

```
348   Vector3D P_O_RailF_G_RailF= *sys.new_Vector3D("P_O_RailF_G_RailF"
        ,0.0,0.0,0.0,"B_RailF");
349       Tensor3D I_RailF = *sys.new_Tensor3D("I_RailF",(ex)0.177926,(
        ex)0.0,(ex)0.0,(ex)0.0,(ex)0.05294,(ex)0.0,(ex)0.0,(ex)0.0,(ex)
        0.177927,"B_RailF"); //Desde origen
350
351           //Ch
352           symbol_numeric  m_Ch = *sys.new_Parameter("m_Ch",13.610528)
        ;
353   Vector3D P_O_Ch_G_Ch= *sys.new_Vector3D("P_O_Ch_G_Ch"
        ,0.0,0.0,0.010159,"B_Ch");
354       Tensor3D I_Ch = *sys.new_Tensor3D("I_Ch",(ex)0.220622,(ex)
        0.0,(ex)0.0,(ex)0.0,(ex)0.242055,(ex)0.0,(ex)0.0,(ex)0.0,(ex)
        0.456189,"B_Ch"); //Desde origen
355
356           //WhsB
357           symbol_numeric  m_WhsB = *sys.new_Parameter("m_WhsB"
        ,9.26935);
358   Vector3D P_O_WhsB_G_WhsB= *sys.new_Vector3D("P_O_WhsB_G_WhsB"
        ,0.0,0.0,0.0,"B_WhsB");
359       Tensor3D I_WhsB = *sys.new_Tensor3D("I_WhsB",(ex)0.119118,(ex
        )0.0,(ex)0.0,(ex)0.0,(ex)0.010442,(ex)0.0,(ex)0.0,(ex)0.0,(ex)
        0.119135,"B_WhsB"); //Desde origen
360           //Desprecio el efecto que el chavetero ejerce en WHS,
        tomando la aproximaci n de que el origen coincide en WHS con
        su centro de masas.
361           //M s informaci n en el documento PROPIEDADES PIEZAS
362
363    //WhsF
364           symbol_numeric  m_WhsF = *sys.new_Parameter("m_WhsF"
        ,9.26935);
365   Vector3D P_O_WhsF_G_WhsF= *sys.new_Vector3D("P_O_WhsF_G_WhsF"
        ,0.0,0.0,0.0,"B_WhsF");
366       Tensor3D I_WhsF = *sys.new_Tensor3D("I_WhsF",(ex)0.119118,(ex
        )0.0,(ex)0.0,(ex)0.0,(ex)0.010442,(ex)0.0,(ex)0.0,(ex)0.0,(ex)
        0.119135,"B_WhsF"); //Desde origen
367
368
369
370
371
372 //
        ********************************************************************************
373 //  Define Frames
374 //
        ********************************************************************************
375
376   Frame * F_RailB = sys.new_Frame("F_RailB","O_RailB","B_RailB");
377   Frame * F_RailF = sys.new_Frame("F_RailF","O_RailF","B_RailF");
378   Frame * F_Ch = sys.new_Frame("F_Ch","O_Ch","B_Ch");
379   Frame * F_WhsB = sys.new_Frame("F_WhsB","O_WhsB","B_WhsB");
380   Frame * F_WhsF = sys.new_Frame("F_WhsF","O_WhsF","B_WhsF");
381
382
```

```
383 //
    ********************************************************************************************
384 //  Define Solids
385 //
    ********************************************************************************************
386   Solid * RailB = sys.new_Solid("RailB","O_RailB","B_RailB","
      m_RailB","P_O_RailB_G_RailB","I_RailB");
387   Solid * RailF = sys.new_Solid("RailF","O_RailF","B_RailF","
      m_RailF","P_O_RailF_G_RailF","I_RailF");
388   Solid * Ch = sys.new_Solid("Ch","O_Ch","B_Ch","m_Ch","P_O_Ch_G_Ch
      ","I_Ch");
389   Solid * WhsB = sys.new_Solid("WhsB","O_WhsB","B_WhsB","m_WhsB","
      P_O_WhsB_G_WhsB","I_WhsB");
390   Solid * WhsF = sys.new_Solid("WhsF","O_WhsF","B_WhsF","m_WhsF","
      P_O_WhsF_G_WhsF","I_WhsF");
391
392 //
    ********************************************************************************************
393 //  Define Objects to Draw
394 //
    ********************************************************************************************
395
396         Drawing3D * D_Structure = sys.new_Drawing3D("D_Structure",
      "O", "xyz1" ,"./solids/structure.osgt");
397         Drawing3D * D_RailB = sys.new_Drawing3D("D_RailB", "O_RailB
      ", "B_RailB" ,"./solids/railset.osgt");
398   Drawing3D * D_RailF = sys.new_Drawing3D("D_RailF", "O_RailF", "
      B_RailF" ,"./solids/railset.osgt");
399   Drawing3D * D_Ch = sys.new_Drawing3D("D_Ch", "O_Ch", "B_Ch" ,"./
      solids/chasis.osgt");
400         Drawing3D * D_WhsB = sys.new_Drawing3D("D_WhsB", "O_WhsB",
      "B_WhsB" ,"./solids/wheelset.osgt");
401         Drawing3D * D_WhsF = sys.new_Drawing3D("D_WhsF", "O_WhsF",
      "B_WhsF" ,"./solids/wheelset.osgt");
402
403
404         // Drawing3D * D_WHSR = sys.new_Drawing3D("D_WHSR", "OWHSR",
       "B_WHSR" ,"./solids/sistema_boogie.osgt");
405         //Drawing3D * D_WhsF = sys.new_Drawing3D("D_WhsF", "O_WhsF
      ", "B_WhsF" ,"./solids/sistema_boogie.osgt");
406
407   Drawing3D * D_BearBL = sys.new_Drawing3D("D_BearBL", "
      C_BearBL_WhsB", "B_WhsB_a" ,"./solids/bear_left.osgt");
408         Drawing3D * D_BearBR = sys.new_Drawing3D("D_BearBR", "
      C_BearBR_WhsB", "B_WhsB_a" ,"./solids/bear_right.osgt");
409   Drawing3D * D_BearFL = sys.new_Drawing3D("D_BearFL", "
      C_BearFL_WhsF", "B_WhsF_a" ,"./solids/bear_left.osgt");
410         Drawing3D * D_BearFR = sys.new_Drawing3D("D_BearFR", "
      C_BearFR_WhsF", "B_WhsF_a" ,"./solids/bear_right.osgt");
411
412
413
414
```

```
415         Drawing3D * D_F_RailB = sys.new_Drawing3D("D_F_RailB","
    F_RailB",0.07);
416         Drawing3D * D_F_RailF = sys.new_Drawing3D("D_F_RailF","
    F_RailF",0.07);
417         Drawing3D * D_F_Ch = sys.new_Drawing3D("D_F_Ch","F_Ch",0.2)
    ;
418         Drawing3D * D_F_WhsB = sys.new_Drawing3D("D_F_WhsB","F_WhsB
    ",0.07);
419         Drawing3D * D_F_WhsF = sys.new_Drawing3D("D_F_WhsF","F_WhsF
    ",0.07);
420
421
422
423   //Vector3D P_O_WhsB_C_BearBL_WhsB = *sys.new_Vector3D("
      P_O_WhsB_C_BearBL_WhsB",0,0.188,0,"B_WhsB_a");
424
425   Vector3D Aux76 = *sys.new_Vector3D("Aux76",0.0,0.0,0.1,"xyz1");
426 ;
427
428   Drawing3D * D_Aux76 = sys.new_Drawing3D("D_Aux76",&Aux76,
      C_BearBL_Ch,0,0,1,0.05);
429
430   //Drawing3D * D_C_BearBL_Ch    = sys.new_Drawing3D("
      D_P_O_Ch_C_BearBL_Ch",&P_O_Ch_C_BearBL_Ch,O_Ch);
431   //Drawing3D * D_P_O_WhsB_C_BearBL_WhsB = sys.new_Drawing3D("
      D_P_O_WhsB_C_BearBL_WhsB",&P_O_WhsB_C_BearBL_WhsB  ,O_WhsB);
432 /*
433
434   Drawing3D * D_O_Ch_STRUCTURE = sys.new_Drawing3D("
      D_O_Ch_STRUCTURE",&X_Ch_STRUCTURE,O_Ch);
435
436
437
438   Drawing3D * D_B_WhsF_K_RF1 = sys.new_Drawing3D("D_B_WhsF_K_RF1",&
      O_Ch_OBEAR_RF,O_WhsF);
439   Drawing3D * D_B_WhsF_K_LF1 = sys.new_Drawing3D("D_B_WhsF_K_LF1",&
      O_Ch_OBEAR_LF,O_WhsF);
440   Drawing3D * D_B_WHSR_K_RR1 = sys.new_Drawing3D("D_B_WHSR_K_RR1",&
      O_Ch_OBEAR_RR,OWHSR);
441   Drawing3D * D_B_WHSR_K_LR1 = sys.new_Drawing3D("D_B_WHSR_K_LR1",&
      O_Ch_OBEAR_LR,OWHSR);
442
443
444   Drawing3D * D_B_O_Ch_K_RF1 = sys.new_Drawing3D("D_B_O_Ch_K_RF1",&
      O_Ch_K_RF1,O_Ch);
445   Drawing3D * D_B_O_Ch_K_LF1 = sys.new_Drawing3D("D_B_O_Ch_K_LF1",&
      O_Ch_K_LF1,O_Ch);
446   Drawing3D * D_B_O_Ch_K_RR1 = sys.new_Drawing3D("D_B_O_Ch_K_RR1",&
      O_Ch_K_RR1,O_Ch);
447   Drawing3D * D_B_O_Ch_K_LR1 = sys.new_Drawing3D("D_B_O_Ch_K_LR1",&
      O_Ch_K_LR1,O_Ch);
448
449
450
451
452
453
```

```
454   Drawing3D * D_O_Ch_K_RF1 = sys.new_Drawing3D("D_O_Ch_K_RF1",&
        O_Ch_K_RF1,O_Ch);
455
456   Drawing3D * D_B_WHSR_K_RR1 = sys.new_Drawing3D("D_B_WHSR_K_RR1",&
        B_WHSR_K_RR1,OWHSR);
457   Drawing3D * D_O_Ch_K_RR1 = sys.new_Drawing3D("D_O_Ch_K_RR1",&
        O_Ch_K_RR1,O_Ch);
458
459         Drawing3D * D_B_WhsF_K_RF2 = sys.new_Drawing3D("
        D_B_WhsF_K_RF2",&B_WhsF_K_RF2,O_WhsF);
460   Drawing3D * D_B_WhsF_K_LF1 = sys.new_Drawing3D("D_B_WhsF_K_LF1",&
        B_WhsF_K_LF1,O_WhsF);
461   Drawing3D * D_B_WhsF_K_LF2 = sys.new_Drawing3D("D_B_WhsF_K_LF2",&
        B_WhsF_K_LF2,O_WhsF);
462
463   Drawing3D * D_O_Ch_K_RF2 = sys.new_Drawing3D("D_O_Ch_K_RF2",&
        O_Ch_K_RF2,O_Ch);
464   Drawing3D * D_O_Ch_K_LF1 = sys.new_Drawing3D("D_O_Ch_K_LF1",&
        O_Ch_K_LF1,O_Ch);
465   Drawing3D * D_O_Ch_K_LF2 = sys.new_Drawing3D("D_O_Ch_K_LF2",&
        O_Ch_K_LF2,O_Ch);
466 */
467
468 //
      ***************************************************************************************
469 //  Joint Unknown Definition
470 //
      ***************************************************************************************
471
472   cout << "Joint Unknown Definition" << endl;
473
474
475
476 //
      ***************************************************************************************
477 //  Input Vector
478 //
      ***************************************************************************************
479
480   cout << "Input Vector" << endl;
481
482   symbol_numeric Fx_BL = *sys.new_Input("Fx_BL",0.0);
483     symbol_numeric Fy_BL = *sys.new_Input("Fy_BL",0.0);
484     symbol_numeric Mz_BL = *sys.new_Input("Mz_BL",0.0);
485
486         symbol_numeric Fx_BR = *sys.new_Input("Fx_BR",0.0);
487     symbol_numeric Fy_BR = *sys.new_Input("Fy_BR",0.0);
488     symbol_numeric Mz_BR = *sys.new_Input("Mz_BR",0.0);
489
490
491       symbol_numeric Fx_FL = *sys.new_Input("Fx_FL",0.0);
492     symbol_numeric Fy_FL = *sys.new_Input("Fy_FL",0.0);
493     symbol_numeric Mz_FL = *sys.new_Input("Mz_FL",0.0);
```

```
494
495
496
497         symbol_numeric Fx_FR = *sys.new_Input("Fx_FR",0.0);
498       symbol_numeric Fy_FR = *sys.new_Input("Fy_FR",0.0);
499       symbol_numeric Mz_FR = *sys.new_Input("Mz_FR",0.0);
500
501
502
503     // Motors
504         symbol_numeric T_MOTF = *sys.new_Input("T_MOTF",0.0);
505         symbol_numeric T_MOTB = *sys.new_Input("T_MOTB",0.0);
506
507     symbol_numeric T_WhsF = *sys.new_Input("T_WhsF",0.0);
508         symbol_numeric T_WhsB  = *sys.new_Input("T_WhsF",0.0);
509
510     //Aditional Forces
511     symbol_numeric F_Ax = *sys.new_Input("F_Ax",0.0);
512     symbol_numeric F_Ay = *sys.new_Input("F_Ay",0.0);
513     symbol_numeric F_Az = *sys.new_Input("F_Az",0.0);
514
515     symbol_numeric M_Ax = *sys.new_Input("M_Ax",0.0);
516     symbol_numeric M_Ay = *sys.new_Input("M_Ay",0.0);
517     symbol_numeric M_Az = *sys.new_Input("M_Az",0.0);
518
519
520
521 //
        ***************************************************************************************
522 //   Force and Momentum Definition
523 //
        ***************************************************************************************
524
525
526     Vector3D F_F_A= *sys.new_Vector3D("F_F_A",F_Ax,F_Ay,F_Az,"B_Ch");
527         Vector3D M_F_A = *sys.new_Vector3D("M_F_A ",M_Ax,M_Ay,M_Az, "
        B_Ch");
528     Vector3D F_MOTF = *sys.new_Vector3D("F_MOTF",0,0,0,"B_RailF");
529         Vector3D M_MOTF = *sys.new_Vector3D("M_MOTF",0,T_MOTF,0, "
        B_RailF");
530         Vector3D F_MOTB = *sys.new_Vector3D("F_MOTB",0,0,0,"B_RailB")
        ;
531         Vector3D M_MOTB = *sys.new_Vector3D("M_MOTB",0,T_MOTB,0, "
        B_RailB");
532
533     Vector3D FT_WhsF = *sys.new_Vector3D("FT_WhsF",0,0,0,"B_WhsF_a");
534         Vector3D M_WhsF = *sys.new_Vector3D("M_WhsF",0,T_WhsF,0, "
        B_WhsF_a");
535         Vector3D FT_WhsB = *sys.new_Vector3D("FT_WhsB",0,0,0,"
        B_WhsB_a");
536         Vector3D M_WhsB = *sys.new_Vector3D("M_WhsB",0,T_WhsB,0, "
        B_WhsB_a");
537
538 //
        ***************************************************************************************
```

```
539 //  Define Wrenches
540 //
    ********************************************************************************

541
542   cout << "Define Wrenches" << endl;
543
544
545
546
547   //  Gravity
548
549   Wrench3D * Gravity_RailB = sys.Gravity_Wrench("RailB");
550   Wrench3D * Gravity_RailF = sys.Gravity_Wrench("RailF");
551   Wrench3D * Gravity_Ch = sys.Gravity_Wrench("Ch");
552   Wrench3D * Gravity_WhsB = sys.Gravity_Wrench("WhsB");
553   Wrench3D * Gravity_WhsF = sys.Gravity_Wrench("WhsF");
554
555
556   //  Inertia
557
558   Wrench3D * Inertia_RailB = sys.Inertia_Wrench("RailB");
559   Wrench3D * Inertia_RailF = sys.Inertia_Wrench("RailF");
560   Wrench3D * Inertia_Ch = sys.Inertia_Wrench("Ch");
561   Wrench3D * Inertia_WhsB = sys.Inertia_Wrench("WhsB");
562   Wrench3D * Inertia_WhsF = sys.Inertia_Wrench("WhsF");
563
564
565   //  Constitutive
566
567
568   //  External
569
570   Wrench3D * MOTF_Torque = sys.new_Wrench3D ("MOTF_Torque", F_MOTF
        , M_MOTF , O_RailF , RailF , "External");
571   Wrench3D * MOTB_Torque = sys.new_Wrench3D ("MOTB_Torque", F_MOTB
        , M_MOTB , O_RailB , RailB , "External");
572   Wrench3D * F_F_A_Torque = sys.new_Wrench3D ("F_F_A_Torque", F_F_A
        , M_F_A , O_Ch , Ch , "External");
573   Wrench3D * WhsF_Torque = sys.new_Wrench3D ("WhsF_Torque", FT_WhsF
        , M_WhsF , O_WhsF , WhsF , "External");
574   Wrench3D * WhsB_Torque = sys.new_Wrench3D ("WhsB_Torque", FT_WhsB
        , M_WhsB , O_WhsB , WhsB , "External");
575
576 //
    ********************************************************************************

577 //  Springs
578 //
    ********************************************************************************

579
580   cout << "springs" << endl;
581
582     symbol_numeric  K_S1xy = *sys.new_Parameter("K_S1xy"
      ,100000.000);
```

```
583     symbol_numeric  K_S1z = *sys.new_Parameter("K_S1z",100000.000);
584
585     symbol_numeric  C_S1xy = *sys.new_Parameter("C_S1xy"
        ,1713.00000000);
586     symbol_numeric  C_S1z = *sys.new_Parameter("C_S1z"
        ,1713.00000000);
587
588
589
590
591
592     symbol_numeric  K_S2 = *sys.new_Parameter("K_S2",100000.000);
593     symbol_numeric  C_S2 = *sys.new_Parameter("C_S2",1713.00000000)
        ;
594
595   Tensor3D K_S1 = *sys.new_Tensor3D("K_S1",(ex)K_S1xy,(ex)0.0,(ex)
      0.0,(ex)0.0,(ex)K_S1xy,(ex)0.0,(ex)0.0,(ex)0.0,(ex)K_S1z,"B_Ch"
      ); //Desde origen
596   Tensor3D C_S1 = *sys.new_Tensor3D("C_S1",(ex)C_S1xy,(ex)0.0,(ex)
      0.0,(ex)0.0,(ex)C_S1xy,(ex)0.0,(ex)0.0,(ex)0.0,(ex)C_S1z,"B_Ch"
      ); //Desde origen
597   //cout << "tensor" << K_S1 <<  endl;
598
599 //Spring BL
600     Vector3D aux_1 = sys.Position_Vector("C_BearBL_Ch","
      C_BearBL_WhsB");
601     Vector3D aux_2 = sys.Dt(sys.Position_Vector("C_BearBL_Ch","
      C_BearBL_WhsB"),"B_WhsB_a");
602     Vector3D F_SBL =  K_S1 * aux_1 + C_S1 * aux_2 ;
603     Vector3D M_SBL   = *sys.new_Vector3D("M_SBL",0,0,0, "B_WhsB_a")
        ;
604
605
606     Wrench3D * TA_SBL= sys.new_Wrench3D ( "TA_SBL",-F_SBL , -M_SBL
        , C_BearBL_WhsB   ,WhsB , "Constitutive") ;
607     Wrench3D * TR_SBL= sys.new_Wrench3D ( "TR_SBL",F_SBL , M_SBL,
        C_BearBL_Ch   ,Ch , "Constitutive") ;
608
609
610 // Spring BR
611     aux_1 = sys.Position_Vector("C_BearBR_Ch","C_BearBR_WhsB");
612     aux_2 = sys.Dt(sys.Position_Vector("C_BearBR_Ch","C_BearBR_WhsB
        "),"B_WhsB_a");
613     Vector3D F_SBR =  K_S1 * aux_1 + C_S1 * aux_2;
614     Vector3D M_SBR   = *sys.new_Vector3D("M_SBR",0,0,0, "B_WhsB_a")
        ;
615
616
617     Wrench3D * TA_SBR = sys.new_Wrench3D ( "TA_SBR",-F_SBR, -M_SBR
        , C_BearBR_WhsB   ,WhsB , "Constitutive") ;
618     Wrench3D * TR_SBR = sys.new_Wrench3D ( "TR_SBR",F_SBR ,M_SBR,
        C_BearBR_Ch ,Ch , "Constitutive") ;
619
620
621 // Spring FL
622
623     aux_1 = sys.Position_Vector("C_BearFL_Ch","C_BearFL_WhsF");
```

```
624     aux_2 = sys.Dt(sys.Position_Vector("C_BearFL_Ch","C_BearFL_WhsF
        "),"B_WhsF_a");
625     Vector3D F_SFL =  K_S1 * aux_1 + C_S1 * aux_2;
626     Vector3D M_SFL   = *sys.new_Vector3D("M_SFL",0,0,0, "B_WhsF_a")
        ;
627
628
629   Wrench3D * TA_SFL= sys.new_Wrench3D ( "TA_SFL",-F_SFL   , -M_SFL ,
        C_BearFL_WhsF    , WhsF , "Constitutive") ;
630   Wrench3D * TR_SFL= sys.new_Wrench3D ( "TR_SFL", F_SFL   , M_SFL ,
        C_BearFL_Ch , Ch , "Constitutive") ;
631
632 // Spring FR
633
634
635     cout << sys.Position_Vector("C_BearFR_Ch","C_BearFR_WhsF") <<
        endl;
636
637
638     aux_1 =sys.Position_Vector("C_BearFR_Ch","C_BearFR_WhsF");
639     aux_2 = sys.Dt(sys.Position_Vector("C_BearFR_Ch","C_BearFR_WhsF
        "),"B_WhsF_a");
640     Vector3D F_SFR = K_S1 * aux_1   +  C_S1 * aux_2;
641     Vector3D M_SFR   = *sys.new_Vector3D("M_SFR ",0,0,0, "B_WhsF_a"
        );
642
643
644   Wrench3D * TA_SFR= sys.new_Wrench3D ( "TA_SFR",-F_SFR , -M_SFR ,
        C_BearFR_WhsF   ,WhsF , "Constitutive") ;
645   Wrench3D * TR_SFR= sys.new_Wrench3D ( "TR_SFR", F_SFR ,M_SFR,
        C_BearFR_Ch   ,Ch , "Constitutive") ;
646
647
648
649
650
651
652
653 // Spring 5
654
655         Vector3D e_x_xyz1 = *sys.new_Vector3D("e_x_xyz1",1,0,0,"
        xyz1");
656         Vector3D mypositionvector = sys.Position_Vector("O","O_Ch")
        ;
657         ex xcompofmypositionvector = mypositionvector * e_x_xyz1;
658   Vector3D e_x_position=*sys.new_Vector3D("e_x_position",
        xcompofmypositionvector,0,0,"B_Ch");
659
660   //Point * OCh_STRUCTURE = sys.new_Point("OCh_STRUCTURE","OCh",&
        e_x_position);
661     //Vector3D F1_SRB1 = K_S2 * sys.Position_Vector("OCh","
        OCh_STRUCTURE") + C_S2 * sys.Dt(sys.Position_Vector("OCh","
        OCh_STRUCTURE"),"B_Ch");
662         Vector3D F_S_O_O_Ch = K_S2 * e_x_position + C_S2 * sys.Dt(
        e_x_position,"xyz1");
663
```

```
664   Vector3D M_S_O_O_Ch  = *sys.new_Vector3D("M_S_O_O_Ch",0,0,0, "
        B_Ch");
665
666   cout<<unatomize( F_S_O_O_Ch)<< endl;
667
668
669   //Wrench3D * TA1_SRR1= sys.new_Wrench3D ( "TA1_SRR1",F1_SRR1 ,
        M_SRR1 , P_OCh_K_RR1 ,WHSR , "Constitutive") ;
670     Wrench3D * TR_S_O_O_Ch = sys.new_Wrench3D ( "TR_S_O_O_Ch",-
        F_S_O_O_Ch ,-M_S_O_O_Ch, O_Ch ,Ch , "Constitutive") ;
671
672
673
674 //
      ********************************************************************************
675 //    Kinematic Equations
676 //
      ********************************************************************************
677   cout << "Kinematic Equations" << endl;
678         Matrix Phi(20,1);
679
680
681         symbol_numeric RRAIL = *sys.new_Parameter("RRAIL",0.10625);
       //radio_RAIL
682         symbol_numeric RWHS = *sys.new_Parameter("RWHS",0.053125);
      //radio_WHS
683         symbol_numeric LRAIL = *sys.new_Parameter("LRAIL",0.2185);
      //distancia_RAIL("LRAIL",0.23184);
684         symbol_numeric LWHS = *sys.new_Parameter("LWHS",0.2185); //
      distancia_WHS
685
686         //BACK LEFT
687   Vector3D P_O_WhsB_O_WhsBL = *sys.new_Vector3D("P_O_WhsB_O_WhsBL"
        ,0,0.5*LWHS,0,"B_WhsB_a");
688         Point * O_WhsBL = sys.new_Point("O_WhsBL","O_WhsB",&
      P_O_WhsB_O_WhsBL);
689
690   symbol_numeric thetaW_BL = *sys.new_Coordinate("thetaW_BL","
        dthetaW_BL","ddthetaW_BL", 3.1416, 0.0 , 0.0);
691   symbol_numeric uW_BL = *sys.new_Coordinate("uW_BL","duW_BL","
        dduW_BL", 0.0 , 0.0 , 0.0);
692
693         symbol_numeric aW_BL = *sys.new_Parameter("aW_BL"
      ,0.000000000);
694         symbol_numeric bW_BL = *sys.new_Parameter("bW_BL"
      ,0.000000000);
695         symbol_numeric cW_BL = *sys.new_Parameter("cW_BL",0.1820);
      //*sys.new_Parameter("cW_RL",2.7049);
696         symbol_numeric dW_BL = *sys.new_Parameter("dW_BL"
      ,-0.053125); //*sys.new_Parameter("dW_RL",0.053125);
697         symbol_numeric lW_BL = *sys.new_Parameter("lW_BL"
      ,0.0000000000);
698
699         ex sW_BL = uW_BL- lW_BL; //;
```

```
700         ex splineW_BL = dW_BL + sW_BL * ( cW_BL + sW_BL * (bW_BL +
    sW_BL * aW_BL ) );
701         Vector3D fW_BL = *sys.new_Vector3D("fW_BL",splineW_BL*sin(
    thetaW_BL),uW_BL,splineW_BL*cos(thetaW_BL),"B_WhsB_a");
702         Vector3D rW_BL = sys.Position_Vector("O","O_WhsBL")+fW_BL;
703         Vector3D t1W_BL= sys.diff ( rW_BL , thetaW_BL);
704         Vector3D t2W_BL= sys.diff ( rW_BL , uW_BL);
705         Vector3D nW_BL = t1W_BL^t2W_BL;
706
707
708 Vector3D P_O_RailB_O_RailBL = *sys.new_Vector3D("
    P_O_RailB_O_RailBL",0,0.5*LRAIL,0,"B_RailB");
709         Point * O_RailBL = sys.new_Point("O_RailBL","O_RailB",&
    P_O_RailB_O_RailBL);
710
711 symbol_numeric uR_BL = *sys.new_Coordinate("uR_BL","duR_BL","
    dduR_BL", 0.0 , 0.0 , 0.0);
712 symbol_numeric thetaR_BL = *sys.new_Coordinate("thetaR_BL","
    dthetaR_BL","ddthetaR_BL", 0.0 , 0.0 , 0.0);
713
714         symbol_numeric aR_BL = *sys.new_Parameter("aR_BL"
    ,0.00000000000);
715         symbol_numeric bR_BL = *sys.new_Parameter("bR_BL",-2.7473);
     //*sys.new_Parameter("bR_RL",-2.7049);
716         symbol_numeric cR_BL = *sys.new_Parameter("cR_BL"
    ,0.00000000000);
717         symbol_numeric dR_BL = *sys.new_Parameter("dR_BL",0.10625);
     //*sys.new_Parameter("dR_RL",0.10625);
718         symbol_numeric lR_BL = *sys.new_Parameter("lR_BL"
    ,0.00000000000);
719
720         ex sR_BL = uR_BL- lR_BL; //;
721         ex splineR_BL =  dR_BL + sR_BL * ( cR_BL + sR_BL * (bR_BL +
     sR_BL * aR_BL ) );
722         Vector3D fR_BL = *sys.new_Vector3D("fR_BL",splineR_BL*sin(
    thetaR_BL),uR_BL,splineR_BL*cos(thetaR_BL),"xyz1"); //*sys.
    new_Vector3D("fR_RL",thetaR_RL,uR_RL,splineR_RL,"B_RAILR");
723         Vector3D rR_BL = sys.Position_Vector("O","O_RailBL")+fR_BL;
724         Vector3D t1R_BL= sys.diff ( rR_BL , thetaR_BL);
725         Vector3D t2R_BL= sys.diff ( rR_BL , uR_BL);
726         Vector3D nR_BL = t1R_BL^t2R_BL;
727
728         //normal and tangent unitary vectors at contact point
729         Vector3D u_nR_BL =( 1 /sqrt( nR_BL * nR_BL )) * nR_BL;
730       Vector3D u_t1R_BL =( 1 /sqrt( t1R_BL * t1R_BL )) * t1R_BL;
731       Vector3D u_t2R_BL =( 1 /sqrt( t2R_BL * t2R_BL )) * t2R_BL;
732
733
734     Point * J_BL_Wheel    = sys.new_Point("J_BL_Wheel","O",&rW_BL)
    ;
735         Point * J_BL_Rail = sys.new_Point("J_BL_Rail","O",&rR_BL);
736
737
738 Drawing3D * D_J_BL_Wheel = sys.new_Drawing3D("D_J_BL_Wheel",
    J_BL_Wheel,0.04);
739
```

```
740   Drawing3D * D_u_nR_BL  = sys.new_Drawing3D("D_u_nR_BL",&u_nR_BL,
        J_BL_Rail);
741   Drawing3D * D_u_t1R_BL  = sys.new_Drawing3D("D_u_t1R_BL",&
        u_t1R_BL,J_BL_Rail);
742   Drawing3D * D_u_t2R_BL  = sys.new_Drawing3D("D_u_t2R_BL",&
        u_t2R_BL,J_BL_Rail);
743
744     //Vector3D Pcon_BL_W_Rail = sys.Position_Vector("Pcon_BL","
        J_BL_Rail");
745   //Vector3D Vcon_BL_W_Rail=sys.Dt(sys.Position_Vector("Pcon_BL","
        J_BL_Rail"),"xyz1");
746
747         Vector3D V_abs_J_BL_Wheel= sys.Velocity_Vector ("abs" , "
        J_BL_Wheel", "WhsB" );
748         Vector3D V_abs_Pcon_BL_Rail= sys.Velocity_Vector ("abs" , "
        J_BL_Rail", "RailB" );
749         Vector3D Vcon_BL_W_Rail=V_abs_J_BL_Wheel-V_abs_Pcon_BL_Rail
        ;
750
751   Drawing3D * D_Vcon_BL_W_Rail = sys.new_Drawing3D("
        D_Vcon_BL_W_Rail",&Vcon_BL_W_Rail,J_BL_Wheel);
752
753 cout << "V_abs_J_BL_Wheel= " << unatomize(V_abs_J_BL_Wheel) << endl
        ;
754 cout << "V_abs_Pcon_BL_Rail= " << unatomize(V_abs_Pcon_BL_Rail) <<
        endl;
755 cout << "u_t1R_BL= " << unatomize(u_t1R_BL) << endl;
756
757
758   Vector3D Vprom_BL_W_RailAux1= V_abs_J_BL_Wheel+V_abs_Pcon_BL_Rail
        ;
759
760   ex Vprom_BL_W_RailAux2 =abs(sqrt( Vprom_BL_W_RailAux1 *
        Vprom_BL_W_RailAux1 ));
761
762   ex Vprom_BL_W_Rail=((numeric(1,2))*Vprom_BL_W_RailAux2);
763
764   //ex Vprom_BL_W_RailAux3 =sqrt( Vprom_BL_W_RailAux1 *
        Vprom_BL_W_RailAux1 );
765   ex Vprom_BL_W_Rail2=((numeric(1,2))*((V_abs_J_BL_Wheel+
        V_abs_Pcon_BL_Rail)*u_t1R_BL));
766
767   ex Vcreepx_BL= (Vcon_BL_W_Rail*u_t1R_BL);
768   ex Vcreepy_BL= (Vcon_BL_W_Rail*u_t2R_BL);
769
770   Vector3D Vcreepphi_BLAux1=sys.Angular_Velocity("xyz1","B_WhsB");
771   Vector3D Vcreepphi_BLAux2=sys.Angular_Velocity("xyz1","B_RailB");
772
773   Vector3D Vcreepphi_BLAux3= Vcreepphi_BLAux1-Vcreepphi_BLAux2;
774
775   ex Vcreepphi_BL=(Vcreepphi_BLAux3* u_nR_BL);
776
777 /*
778
779   ex V_abs_CBL_wheel= abs((sys.Velocity_Vector ("abs" , "O_WhsBL" )
        )* u_t1R_BL);//ex V_abs_CBL_wheel= abs((sys.Velocity_Vector ("
        abs" , "O_WhsBL" )-V_abs_Pcon_BL_Rail)* u_t1R_BL)
```

```
780   Vector3D Aux1_angularBL=(sys.Angular_Velocity ("xyz1","B_WhsB"));
781   Vector3D Aux2_angularBL= sys.Position_Vector("O_WhsBL","
        J_BL_Wheel");
782
783   //ex Aux3_angularBL=(Aux2_angularBL) * (u_t1R_BL);
784   ex Aux_angularBL= (Aux1_angularBL ^ Aux2_angularBL)* u_t1R_BL;
785
786   ex angularBL=abs(Aux_angularBL);
787
788
789   ex Vcreepx_BL= (Vcon_BL_W_Rail*u_t1R_BL)/((numeric(1,2))*(
        V_abs_CBL_wheel+ angularBL)));
790   ex Vcreepy_BL= (Vcon_BL_W_Rail*u_t2R_BL)/((numeric(1,2))*(
        V_abs_CBL_wheel+ angularBL)));
791   ex Vcreepphi_BL=(sys.Angular_Velocity("xyz1","B_WhsB") * u_nR_BL)
        /((numeric(1,2)*( V_abs_CBL_wheel+angularBL)));
792
793
794 */
795   cout<<unatomize(Vcon_BL_W_Rail)<< endl;
796
797       Vector3D FKALKER_BL = (Fx_BL* u_t1R_BL) + (Fy_BL * u_t2R_BL);
798       Vector3D MKALKER_BL = Mz_BL * u_nR_BL;
799
800         Drawing3D * D_FKALKER_BL= sys.new_Drawing3D("D_FKALKER_BL"
      ,&FKALKER_BL, J_BL_Rail,0,1,0,0.5);
801
802       Wrench3D * FKalkerA_BL = sys.new_Wrench3D ( "FKalkerA_BL",
      FKALKER_BL, MKALKER_BL , J_BL_Wheel, WhsB , "Constitutive") ;
803         Wrench3D * FKalkerR_BL = sys.new_Wrench3D ( "FKalkerR_BL",-
      FKALKER_BL,- MKALKER_BL , J_BL_Rail, RailB , "Constitutive") ;
804
805       Phi(0,0) = u_t1R_BL * (rW_BL-rR_BL);
806       Phi(1,0) = u_t2R_BL * (rW_BL-rR_BL);
807       Phi(2,0) = u_nR_BL  * (rW_BL-rR_BL);
808       Phi(3,0) =u_nR_BL*t1W_BL;// nR_RL*t1W_RL
809       Phi(4,0) =u_nR_BL*t2W_BL; //nR_RL*t2W_RL;
810
811     symbol_numeric lambda1  = *sys.new_Joint_Unknown("lambda1");
812       symbol_numeric lambda2  = *sys.new_Joint_Unknown("lambda2");
813         symbol_numeric lambda3  = *sys.new_Joint_Unknown("lambda3")
      ;
814         symbol_numeric lambda4  = *sys.new_Joint_Unknown("lambda4")
      ;
815         symbol_numeric lambda5  = *sys.new_Joint_Unknown("lambda5")
      ;
816
817
818
819         //BACK RIGHT
820
821
822   Vector3D P_O_WhsB_O_WhsBR = *sys.new_Vector3D("P_O_WhsB_O_WhsBR"
      ,0,-0.5*LWHS,0,"B_WhsB_a");
823         Point * O_WhsBR = sys.new_Point("O_WhsBR","O_WhsB",&
      P_O_WhsB_O_WhsBR);
824
```

```
825  symbol_numeric thetaW_BR = *sys.new_Coordinate("thetaW_BR","
       dthetaW_BR","ddthetaW_BR", 3.1416 , 0.0 , 0.0);
826  symbol_numeric uW_BR = *sys.new_Coordinate("uW_BR","duW_BR","
       dduW_BR", 0.0 , 0.0 , 0.0);
827
828      symbol_numeric aW_BR = *sys.new_Parameter("aW_BR"
     ,0.0000000000);
829      symbol_numeric bW_BR = *sys.new_Parameter("bW_BR"
     ,0.0000000000);
830      symbol_numeric cW_BR = *sys.new_Parameter("cW_BR",0.1820);
     //*sys.new_Parameter("cW_RR",-2.7049)
831      symbol_numeric dW_BR = *sys.new_Parameter("dW_BR"
     ,-0.053125); //*sys.new_Parameter("dW_RR",-0.053125);
832      symbol_numeric lW_BR = *sys.new_Parameter("lW_BR"
     ,0.0000000000);
833
834      ex sW_BR = uW_BR- lW_BR; //;
835      ex splineW_BR =  dW_BR + sW_BR * ( cW_BR + sW_BR * (bW_BR +
      sW_BR * aW_BR ) );
836      Vector3D fW_BR = *sys.new_Vector3D("fW_BR",splineW_BR*sin(
     thetaW_BR),-uW_BR,splineW_BR*cos(thetaW_BR),"B_WhsB_a");
837      Vector3D rW_BR = sys.Position_Vector("O","O_WhsBR")+fW_BR;
838      Vector3D t1W_BR= sys.diff ( rW_BR , thetaW_BR);
839      Vector3D t2W_BR= -(sys.diff ( rW_BR , uW_BR));
840      Vector3D nW_BR = t1W_BR^t2W_BR;
841
842
843  Vector3D P_O_RailB_ORailBR = *sys.new_Vector3D("P_O_RailB_ORailBR
     ",0,-0.5*LRAIL,0,"B_RailB");
844      Point * O_RailBR = sys.new_Point("O_RailBR","O_RailB",&
     P_O_RailB_ORailBR);
845
846  symbol_numeric uR_BR = *sys.new_Coordinate("uR_BR","duR_BR","
       dduR_BR", 0.0 , 0.0 , 0.0);
847  symbol_numeric thetaR_BR = *sys.new_Coordinate("thetaR_BR","
       dthetaR_BR","ddthetaR_BR", 0.0 , 0.0 , 0.0);
848
849      symbol_numeric aR_BR = *sys.new_Parameter("aR_BR"
     ,0.0000000000);
850      symbol_numeric bR_BR = *sys.new_Parameter("bR_BR",-2.7473);
      //*sys.new_Parameter("bR_RR",-2.7049);
851      symbol_numeric cR_BR = *sys.new_Parameter("cR_BR"
     ,0.0000000000);
852      symbol_numeric dR_BR = *sys.new_Parameter("dR_BR",0.10625);
      //*sys.new_Parameter("dR_RR",0.10625);
853      symbol_numeric lR_BR = *sys.new_Parameter("lR_BR"
     ,0.0000000000);
854
855      ex sR_BR = uR_BR - lR_BR;//;
856      ex splineR_BR =   dR_BR + sR_BR * ( cR_BR + sR_BR * (bR_BR
     + sR_BR * aR_BR ) );
857      Vector3D fR_BR = *sys.new_Vector3D("fR_BR",splineR_BR*sin(
     thetaR_BR),-uR_BR,splineR_BR*cos(thetaR_BR),"xyz1");
858      Vector3D rR_BR = sys.Position_Vector("O","O_RailBR")+fR_BR;
       //*sys.new_Vector3D("fR_RR",thetaR_RR,uR_RR,splineR_RR,"
     B_RAILR");
859      Vector3D t1R_BR= sys.diff ( rR_BR , thetaR_BR);
```

```
860         Vector3D t2R_BR= -(sys.diff ( rR_BR , uR_BR));
861         Vector3D nR_BR = t1R_BR^t2R_BR;
862
863         //normal and tangent unitary vectors at contact point
864         Vector3D u_nR_BR =( 1 /sqrt( nR_BR * nR_BR )) * nR_BR;
865       Vector3D u_t1R_BR =( 1 /sqrt( t1R_BR * t1R_BR )) * t1R_BR;
866       Vector3D u_t2R_BR =( 1 /sqrt( t2R_BR * t2R_BR )) * t2R_BR;
867
868
869       Point * J_BR_Wheel   = sys.new_Point("J_BR_Wheel","O",&rW_BR)
      ;
870         Point * J_BR_RailB = sys.new_Point("J_BR_RailB","O",&rR_BR)
      ;
871
872   Drawing3D * D_J_BR_Wheel = sys.new_Drawing3D("D_J_BR_Wheel",
      J_BR_Wheel,0.04);
873
874   Drawing3D * D_u_nR_BR  = sys.new_Drawing3D("D_u_nR_BR",&u_nR_BR,
      J_BR_RailB);
875   Drawing3D * D_u_t1R_BR  = sys.new_Drawing3D("D_u_t1R_BR",&
      u_t1R_BR,J_BR_RailB);
876   Drawing3D * D_u_t2R_BR  = sys.new_Drawing3D("D_u_t2R_BR",&
      u_t2R_BR,J_BR_RailB);
877
878   //Vector3D Pcon_BR_W_Rail = sys.Position_Vector("Pcon_BR","
      J_BR_RailB");
879   //Vector3D Vcon_BR_W_Rail=sys.Dt(sys.Position_Vector("Pcon_BR","
      J_BR_RailB"),"xyz1");
880
881         Vector3D V_abs_J_BR_Wheel= sys.Velocity_Vector ("abs" , "
      J_BR_Wheel", "WhsB" );
882         Vector3D V_abs_Pcon_BR_Rail= sys.Velocity_Vector ("abs" , "
      J_BR_RailB", "RailB" );
883         Vector3D Vcon_BR_W_Rail=V_abs_J_BR_Wheel-V_abs_Pcon_BR_Rail
      ;
884
885   Drawing3D * D_Vcon_BR_W_Rail = sys.new_Drawing3D("
      D_Vcon_BR_W_Rail",&Vcon_BR_W_Rail,J_BR_RailB);
886
887   cout<<unatomize(Vcon_BR_W_Rail)<< endl;
888
889   Vector3D Vprom_BR_W_RailAux1= V_abs_J_BR_Wheel+V_abs_Pcon_BR_Rail
      ;
890
891   ex Vprom_BR_W_RailAux2 =abs(sqrt( Vprom_BR_W_RailAux1 *
      Vprom_BR_W_RailAux1 ));
892
893   ex Vprom_BR_W_Rail=((numeric(1,2))*Vprom_BR_W_RailAux2);
894
895   ex Vprom_BR_W_Rail2=((numeric(1,2))*((V_abs_J_BR_Wheel+
      V_abs_Pcon_BR_Rail)*u_t1R_BR));
896
897   ex Vcreepx_BR= (Vcon_BR_W_Rail*u_t1R_BR);
898   ex Vcreepy_BR= (Vcon_BR_W_Rail*u_t2R_BR);
899
900   Vector3D Vcreepphi_BRAux1=sys.Angular_Velocity("xyz1","B_WhsB");
901   Vector3D Vcreepphi_BRAux2=sys.Angular_Velocity("xyz1","B_RailB");
```

```
902
903   Vector3D Vcreepphi_BRAux3= Vcreepphi_BRAux1-Vcreepphi_BRAux2;
904
905   ex Vcreepphi_BR=(Vcreepphi_BRAux3* u_nR_BR);
906
907
908
909 /*
910   ex V_abs_CBR_wheel= abs((sys.Velocity_Vector ("abs" , "O_WhsBR" )
        )* u_t1R_BR);
911   Vector3D Aux1_angularBR=(sys.Angular_Velocity ("xyz1","B_WhsB"));
912   Vector3D Aux2_angularBR= sys.Position_Vector("O_WhsBR","
        J_BR_Wheel");
913
914
915   ex Aux_angularBR= (Aux1_angularBR ^ Aux2_angularBR)* u_t1R_BR;
916
917
918   ex angularBR=abs(Aux_angularBR);
919
920   ex Vcreepx_BR= (Vcon_BR_W_Rail*u_t1R_BR)/((numeric(1,2)*(
        V_abs_CBR_wheel+ angularBR)));
921   ex Vcreepy_BR= (Vcon_BR_W_Rail*u_t2R_BR)/((numeric(1,2)*(
        V_abs_CBR_wheel+ angularBR)));
922   ex Vcreepphi_BR=(sys.Angular_Velocity("xyz1","B_WhsB") * u_nR_BR)
        /((numeric(1,2)*( V_abs_CBR_wheel+angularBR)));
923
924 */
925   Vector3D FKALKER_BR = (Fx_BR* u_t1R_BR) + (Fy_BR * u_t2R_BR);
926       Vector3D MKALKER_BR = Mz_BR * u_nR_BR;
927
928   Drawing3D * D_FKALKER_BR = sys.new_Drawing3D("D_FKALKER_BR",&
        FKALKER_BR, J_BR_RailB ,0 ,1 ,0 ,0.5);
929
930       Wrench3D * FKalkerA_BR = sys.new_Wrench3D ( "FKalkerA_BR",
        FKALKER_BR, MKALKER_BR , J_BR_Wheel, WhsB , "Constitutive") ;
931       Wrench3D * FKalkerR_BR = sys.new_Wrench3D ( "FKalkerR_BR",-
        FKALKER_BR,- MKALKER_BR ,J_BR_RailB, RailB , "Constitutive") ;
932
933       Phi(5,0) = u_t1R_BR * (rW_BR-rR_BR);
934       Phi(6,0) = u_t2R_BR * (rW_BR-rR_BR);
935       Phi(7,0) = u_nR_BR  * (rW_BR-rR_BR);
936       Phi(8,0) = u_nR_BR*t1W_BR;//nR_RR*t1W_RR;
937       Phi(9,0) = u_nR_BR*t2W_BR;//nR_RR*t2W_RR
938
939         symbol_numeric lambda6  = *sys.new_Joint_Unknown("lambda6")
        ;
940         symbol_numeric lambda7  = *sys.new_Joint_Unknown("lambda7")
        ;
941         symbol_numeric lambda8  = *sys.new_Joint_Unknown("lambda8")
        ;
942         symbol_numeric lambda9  = *sys.new_Joint_Unknown("lambda9")
        ;
943     symbol_numeric lambda10  = *sys.new_Joint_Unknown("lambda10");
944
945 //----
946         //FRONT LEFT
```

```
947
948   Vector3D P_O_WhsF_O_WhsFL = *sys.new_Vector3D("P_O_WhsF_O_WhsFL"
        ,0,0.5*LWHS,0,"B_WhsF_a");
949       Point * O_WhsFL = sys.new_Point("O_WhsFL","O_WhsF",&
      P_O_WhsF_O_WhsFL);
950
951
952   symbol_numeric thetaW_FL = *sys.new_Coordinate("thetaW_FL","
      dthetaW_FL","ddthetaW_FL", 3.1416   , 0.0 , 0.0);
953   symbol_numeric uW_FL = *sys.new_Coordinate("uW_FL","duW_FL","
      dduW_FL", 0.0 , 0.0 , 0.0);
954
955       symbol_numeric aW_FL = *sys.new_Parameter("aW_FL"
      ,0.000000000);
956       symbol_numeric bW_FL = *sys.new_Parameter("bW_FL"
      ,0.000000000);
957       symbol_numeric cW_FL = *sys.new_Parameter("cW_FL",0.1820);
      //*sys.new_Parameter("cW_FL",2.7049);
958       symbol_numeric dW_FL = *sys.new_Parameter("dW_FL"
      ,-0.053125); // *sys.new_Parameter("dW_FL",-0.053125);
959       symbol_numeric lW_FL = *sys.new_Parameter("lW_FL"
      ,0.000000000);
960
961       ex sW_FL = uW_FL- lW_FL ;//;
962       ex splineW_FL =  dW_FL + sW_FL * ( cW_FL + sW_FL * (bW_FL +
       sW_FL * aW_FL ) );
963       Vector3D fW_FL = *sys.new_Vector3D("fW_FL",splineW_FL*sin(
      thetaW_FL),uW_FL,splineW_FL*cos(thetaW_FL),"B_WhsF_a");
964       Vector3D rW_FL = sys.Position_Vector("O","O_WhsFL")+fW_FL;
965       Vector3D t1W_FL= sys.diff ( rW_FL , thetaW_FL);
966       Vector3D t2W_FL= sys.diff ( rW_FL , uW_FL);
967       Vector3D nW_FL = t1W_FL^t2W_FL;
968
969
970   Vector3D P_O_RailF_O_RailFL = *sys.new_Vector3D("
      P_O_RailF_O_RailFL",0,0.5*LRAIL,0,"B_RailF");
971       Point * O_RailFL = sys.new_Point("O_RailFL","O_RailF",&
      P_O_RailF_O_RailFL);
972
973   symbol_numeric uR_FL = *sys.new_Coordinate("uR_FL","duR_FL","
      dduR_FL", 0.0 , 0.0 , 0.0);
974   symbol_numeric thetaR_FL = *sys.new_Coordinate("thetaR_FL","
      dthetaR_FL","ddthetaR_FL", 0.0   , 0.0 , 0.0);
975
976       symbol_numeric aR_FL = *sys.new_Parameter("aR_FL"
      ,0.000000000);
977       symbol_numeric bR_FL = *sys.new_Parameter("bR_FL",-2.7473);
      //*sys.new_Parameter("bR_FL",-2.7049);
978       symbol_numeric cR_FL = *sys.new_Parameter("cR_FL"
      ,0.000000000);
979       symbol_numeric dR_FL = *sys.new_Parameter("dR_FL",0.10625);
       //*sys.new_Parameter("dR_FL",0.10625);
980       symbol_numeric lR_FL = *sys.new_Parameter("lR_FL"
      ,0.000000000);
981
982       ex sR_FL = uR_FL- lR_FL ;//;
```

```
983         ex splineR_FL =  dR_FL + sR_FL * ( cR_FL + sR_FL * (bR_FL +
     sR_FL * aR_FL ) );
984         Vector3D fR_FL = *sys.new_Vector3D("fR_FL",splineR_FL*sin(
    thetaR_FL),uR_FL,splineR_FL*cos(thetaR_FL),"xyz1") ; //;*sys.
    new_Vector3D("fR_FL",thetaR_FL,sR_FL,splineR_FL,"B_RailF");
985         Vector3D rR_FL = sys.Position_Vector("O","O_RailFL")+fR_FL;
986         Vector3D t1R_FL= sys.diff ( rR_FL , thetaR_FL);
987         Vector3D t2R_FL= sys.diff ( rR_FL , uR_FL);
988         Vector3D nR_FL = t1R_FL^t2R_FL;
989
990         //normal and tangent unitary vectors at contact point
991         Vector3D u_nR_FL =( 1 /sqrt( nR_FL * nR_FL )) * nR_FL;
992       Vector3D u_t1R_FL =( 1 /sqrt( t1R_FL * t1R_FL )) * t1R_FL;
993       Vector3D u_t2R_FL =( 1 /sqrt( t2R_FL * t2R_FL )) * t2R_FL;
994
995
996       Point * J_FL_Wheel    = sys.new_Point("J_FL_Wheel","O",&rW_FL)
    ;
997         Point * J_FL_Rail = sys.new_Point("J_FL_Rail","O",&rR_FL);
998
999   Drawing3D * D_J_FL_Wheel = sys.new_Drawing3D("D_J_FL_Wheel",
    J_FL_Wheel,0.04);
1000
1001  Drawing3D * D_u_nR_FL  = sys.new_Drawing3D("D_u_nR_FL",&u_nR_FL,
    J_FL_Rail);
1002  Drawing3D * D_u_t1R_FL  = sys.new_Drawing3D("D_u_t1R_FL",&
    u_t1R_FL,J_FL_Rail);
1003  Drawing3D * D_u_t2R_FL  = sys.new_Drawing3D("D_u_t2R_FL",&
    u_t2R_FL,J_FL_Rail);
1004
1005
1006  //Vector3D Pcon_FL_W_Rail = sys.Position_Vector("Pcon_FL","
    J_FL_Rail");
1007  //Vector3D Vcon_FL_W_Rail=sys.Dt(sys.Position_Vector("Pcon_FL","
    J_FL_Rail"),"xyz1");
1008
1009
1010
1011  Vector3D V_abs_J_FL_Wheel= sys.Velocity_Vector ("abs" , "
    J_FL_Wheel", "WhsF" );
1012        Vector3D V_abs_Pcon_FL_Rail= sys.Velocity_Vector ("abs" , "
    J_FL_Rail", "RailF" );
1013        Vector3D Vcon_FL_W_Rail=V_abs_J_FL_Wheel-V_abs_Pcon_FL_Rail
    ;
1014
1015  Drawing3D * D_Vcon_FL_W_Rail = sys.new_Drawing3D("
    D_Vcon_FL_W_Rail",&Vcon_FL_W_Rail,J_FL_Rail);
1016  cout<<unatomize(Vcon_FL_W_Rail)<< endl;
1017
1018
1019  Vector3D Vprom_FL_W_RailAux1= V_abs_J_FL_Wheel+V_abs_Pcon_FL_Rail
    ;
1020
1021  ex Vprom_FL_W_RailAux2 =abs(sqrt( Vprom_FL_W_RailAux1 *
    Vprom_FL_W_RailAux1 ));
1022
1023  ex Vprom_FL_W_Rail=((numeric(1,2))*Vprom_FL_W_RailAux2);
```

```
1024
1025    ex Vprom_FL_W_Rail2=((numeric(1,2))*((V_abs_J_FL_Wheel+
          V_abs_Pcon_FL_Rail)*u_t1R_FL));
1026
1027    ex Vcreepx_FL= (Vcon_FL_W_Rail*u_t1R_FL);
1028    ex Vcreepy_FL= (Vcon_FL_W_Rail*u_t2R_FL);
1029
1030    Vector3D Vcreepphi_FLAux1=sys.Angular_Velocity("xyz1","B_WhsF");
1031    Vector3D Vcreepphi_FLAux2=sys.Angular_Velocity("xyz1","B_RailF");
1032
1033    Vector3D Vcreepphi_FLAux3= Vcreepphi_FLAux1-Vcreepphi_FLAux2;
1034
1035    ex Vcreepphi_FL=(Vcreepphi_FLAux3* u_nR_FL);
1036   /*
1037    ex V_abs_CFL_wheel= abs((sys.Velocity_Vector ("abs" , "O_WhsFL" )
          )* u_t1R_FL);//abs((sys.Velocity_Vector ("abs" , "O_WhsFL" )-
          V_abs_Pcon_FL_Rail)* u_t1R_FL)
1038    Vector3D Aux1_angularFL=(sys.Angular_Velocity( "xyz1","B_WhsF"));
1039    Vector3D Aux2_angularFL= sys.Position_Vector("O_WhsFL","
          J_FL_Wheel");
1040
1041
1042
1043    ex Aux_angularFL= (Aux1_angularFL ^ Aux2_angularFL)* u_t1R_FL;
1044
1045    ex angularFL=abs(Aux_angularFL);
1046
1047    ex Vcreepx_FL= (Vcon_FL_W_Rail*u_t1R_FL)/((numeric(1,2)*(
          V_abs_CFL_wheel+ angularFL)));
1048    ex Vcreepy_FL= (Vcon_FL_W_Rail*u_t2R_FL)/((numeric(1,2)*(
          V_abs_CFL_wheel+ angularFL)));
1049    ex Vcreepphi_FL=(sys.Angular_Velocity("xyz1","B_WhsF") * u_nR_FL)
          /((numeric(1,2)*( V_abs_CFL_wheel+angularFL)));
1050  */
1051
1052
1053
1054        Vector3D FKALKER_FL = (Fx_FL* u_t1R_FL) + (Fy_FL * u_t2R_FL);
1055        Vector3D MKALKER_FL = Mz_FL * u_nR_FL;
1056
1057    Drawing3D * D_FKALKER_FL = sys.new_Drawing3D("D_FKALKER_FL",&
          FKALKER_FL, J_FL_Rail,0,1,0,0.5);
1058
1059        Wrench3D * FKalkerA_FL = sys.new_Wrench3D ( "FKalkerA_FL",
          FKALKER_FL, MKALKER_FL , J_FL_Wheel, WhsF , "Constitutive") ;
1060          Wrench3D * FKalkerR_FL = sys.new_Wrench3D ( "FKalkerR_FL",-
          FKALKER_FL,- MKALKER_FL , J_FL_Rail, RailF , "Constitutive");
1061
1062        Phi(10,0) = u_t1R_FL * (rW_FL-rR_FL);
1063        Phi(11,0) = u_t2R_FL * (rW_FL-rR_FL);
1064        Phi(12,0) = u_nR_FL  * (rW_FL-rR_FL);
1065        Phi(13,0) = u_nR_FL*t1W_FL;
1066        Phi(14,0) = u_nR_FL*t2W_FL;
1067
1068
1069
```

```
1070        symbol_numeric lambda11  = *sys.new_Joint_Unknown("lambda11
    ");
1071        symbol_numeric lambda12  = *sys.new_Joint_Unknown("lambda12
    ");
1072        symbol_numeric lambda13  = *sys.new_Joint_Unknown("lambda13
    ");
1073        symbol_numeric lambda14  = *sys.new_Joint_Unknown("lambda14
    ");
1074    symbol_numeric lambda15  = *sys.new_Joint_Unknown("lambda15");
1075
1076        //FRONT RIGHT
1077
1078    Vector3D P_O_WhsF_O_WhsFR = *sys.new_Vector3D("P_O_WhsF_O_WhsFR"
    ,0,-0.5*LWHS,0,"B_WhsF_a");
1079        Point * O_WhsFR = sys.new_Point("O_WhsFR","O_WhsF",&
    P_O_WhsF_O_WhsFR);
1080
1081
1082    symbol_numeric thetaW_FR = *sys.new_Coordinate("thetaW_FR","
    dthetaW_FR","ddthetaW_FR", 3.1416 , 0.0 , 0.0);
1083    symbol_numeric uW_FR = *sys.new_Coordinate("uW_FR","duW_FR","
    dduW_FR", 0.0 , 0.0 , 0.0);
1084
1085        symbol_numeric aW_FR = *sys.new_Parameter("aW_FR"
    ,0.0000000000);
1086        symbol_numeric bW_FR = *sys.new_Parameter("bW_FR"
    ,0.0000000000);
1087        symbol_numeric cW_FR = *sys.new_Parameter("cW_FR",0.1820);
    //*sys.new_Parameter("cW_RR",-2.7049)
1088        symbol_numeric dW_FR = *sys.new_Parameter("dW_FR"
    ,-0.053125); //*sys.new_Parameter("dW_RR",-0.053125);
1089        symbol_numeric lW_FR = *sys.new_Parameter("lW_FR"
    ,0.0000000000);
1090
1091        ex sW_FR = uW_FR- lW_FR; //;
1092        ex splineW_FR =  dW_FR + sW_FR * ( cW_FR + sW_FR * (bW_FR +
     sW_FR * aW_FR ) );
1093        Vector3D fW_FR = *sys.new_Vector3D("fW_FR",splineW_FR*sin(
    thetaW_FR),-uW_FR,splineW_FR*cos(thetaW_FR),"B_WhsF_a");
1094        Vector3D rW_FR = sys.Position_Vector("O","O_WhsFR")+fW_FR;
1095        Vector3D t1W_FR= sys.diff ( rW_FR , thetaW_FR);
1096        Vector3D t2W_FR= -(sys.diff ( rW_FR , uW_FR));
1097        Vector3D nW_FR = t1W_FR^t2W_FR;
1098
1099
1100    Vector3D P_O_RailF_O_RailFR  = *sys.new_Vector3D("
    P_O_RailF_O_RailFR ",0,-0.5*LRAIL,0,"B_RailF");
1101        Point * O_RailFR  = sys.new_Point("O_RailFR ","O_RailF",&
    P_O_RailF_O_RailFR );
1102
1103    symbol_numeric uR_FR = *sys.new_Coordinate("uR_FR","duR_FR","
    dduR_FR", 0.0 , 0.0 , 0.0);
1104    symbol_numeric thetaR_FR = *sys.new_Coordinate("thetaR_FR","
    dthetaR_FR","ddthetaR_FR", 0.0 , 0.0 , 0.0);
1105
1106        symbol_numeric aR_FR = *sys.new_Parameter("aR_FR"
    ,0.000000000);
```

```
1107        symbol_numeric bR_FR = *sys.new_Parameter("bR_FR",-2.7473);
     //*sys.new_Parameter("FR_RR",-2.7049);
1108        symbol_numeric cR_FR = *sys.new_Parameter("cR_FR"
    ,0.000000000);
1109        symbol_numeric dR_FR = *sys.new_Parameter("dR_FR",0.10625);
     //*sys.new_Parameter("dR_RR",0.10625);
1110        symbol_numeric lR_FR = *sys.new_Parameter("lR_FR"
    ,0.000000000);
1111
1112        ex sR_FR = uR_FR - lR_FR;//;
1113        ex splineR_FR =   dR_FR + sR_FR * ( cR_FR + sR_FR * (bR_FR
    + sR_FR * aR_FR ) );
1114        Vector3D fR_FR = *sys.new_Vector3D("fR_FR",splineR_FR*sin(
    thetaR_FR),-uR_FR,splineR_FR*cos(thetaR_FR),"xyz1");
1115        Vector3D rR_FR = sys.Position_Vector("O","O_RailFR ")+fR_FR
    ;  //*sys.new_Vector3D("fR_RR",thetaR_RR,uR_RR,splineR_RR,"
    B_RAILR");
1116        Vector3D t1R_FR= sys.diff ( rR_FR , thetaR_FR);
1117        Vector3D t2R_FR= -(sys.diff ( rR_FR , uR_FR));
1118        Vector3D nR_FR = t1R_FR^t2R_FR;
1119
1120        //normal and tangent unitary vectors at contact point
1121        Vector3D u_nR_FR =( 1 /sqrt( nR_FR * nR_FR )) * nR_FR;
1122      Vector3D u_t1R_FR =( 1 /sqrt( t1R_FR * t1R_FR )) * t1R_FR;
1123      Vector3D u_t2R_FR =( 1 /sqrt( t2R_FR * t2R_FR )) * t2R_FR;
1124
1125
1126    Point * J_FR_Wheel   = sys.new_Point("J_FR_Wheel","O",&rW_FR)
    ;
1127        Point * J_FR_Rail = sys.new_Point("J_FR_Rail","O",&rR_FR);
1128
1129
1130  Drawing3D * D_J_FR_Wheel = sys.new_Drawing3D("D_J_FR_Wheel",
    J_FR_Wheel,0.04);
1131
1132  Drawing3D * D_u_nR_FR  = sys.new_Drawing3D("D_u_nR_FR",&u_nR_FR,
    J_FR_Rail);
1133  Drawing3D * D_u_t1R_FR  = sys.new_Drawing3D("D_u_t1R_FR",&
    u_t1R_FR,J_FR_Rail);
1134  Drawing3D * D_u_t2R_FR  = sys.new_Drawing3D("D_u_t2R_FR",&
    u_t2R_FR,J_FR_Rail);
1135
1136  //Vector3D Pcon_FR_W_Rail = sys.Position_Vector("Pcon_FR","
    J_FR_Rail");
1137  //Vector3D Vcon_FR_W_Rail=sys.Dt(sys.Position_Vector("Pcon_FR","
    J_FR_Rail"),"xyz1");
1138
1139        Vector3D V_abs_J_FR_Wheel= sys.Velocity_Vector ("abs" , "
    J_FR_Wheel", "WhsF" );
1140        Vector3D V_abs_Pcon_FR_Rail= sys.Velocity_Vector ("abs" , "
    J_FR_Rail", "RailF" );
1141        Vector3D Vcon_FR_W_Rail=V_abs_J_FR_Wheel-V_abs_Pcon_FR_Rail
    ;
1142
1143  Drawing3D * D_Vcon_FR_W_Rail = sys.new_Drawing3D("
    D_Vcon_FR_W_Rail",&Vcon_FR_W_Rail,J_FR_Rail);
1144
```

```
1145    cout<<unatomize(u_t1R_FR)<< endl;
1146
1147    Vector3D Vprom_FR_W_RailAux1= V_abs_J_FR_Wheel+V_abs_Pcon_FR_Rail
          ;
1148
1149    ex Vprom_FR_W_RailAux2 =abs(sqrt( Vprom_FR_W_RailAux1 *
          Vprom_FR_W_RailAux1 ));
1150
1151    ex Vprom_FR_W_Rail=((numeric(1,2))*Vprom_FR_W_RailAux2);
1152
1153    ex Vprom_FR_W_Rail2=((numeric(1,2))*((V_abs_J_FR_Wheel+
          V_abs_Pcon_FR_Rail)*u_t1R_FR));
1154
1155    ex Vcreepx_FR= (Vcon_FR_W_Rail*u_t1R_FR);
1156    ex Vcreepy_FR= (Vcon_FR_W_Rail*u_t2R_FR);
1157
1158    Vector3D Vcreepphi_FRAux1=sys.Angular_Velocity("xyz1","B_WhsF");
1159    Vector3D Vcreepphi_FRAux2=sys.Angular_Velocity("xyz1","B_RailF");
1160
1161    Vector3D Vcreepphi_FRAux3= Vcreepphi_FRAux1-Vcreepphi_FRAux2;
1162
1163    ex Vcreepphi_FR=(Vcreepphi_FRAux3* u_nR_FR);
1164
1165
1166
1167 /*
1168    ex V_abs_CFR_wheel= abs((sys.Velocity_Vector ("abs" , "O_WhsFR" )
          )* u_t1R_FR);
1169    Vector3D Aux1_angularFR=(sys.Angular_Velocity ("xyz1","B_WhsF"));
1170    Vector3D Aux2_angularFR= sys.Position_Vector("O_WhsFR","
          J_FR_Wheel");
1171
1172
1173
1174    ex Aux_angularFR= (Aux1_angularFR ^ Aux2_angularFR)* u_t1R_FR;
1175
1176
1177    ex angularFR=abs(Aux_angularFR);
1178
1179    ex Vcreepx_FR= (Vcon_FR_W_Rail*u_t1R_FR)/((numeric(1,2)*(
          V_abs_CFR_wheel+ angularFR)));
1180    ex Vcreepy_FR= (Vcon_FR_W_Rail*u_t2R_FR)/((numeric(1,2)*(
          V_abs_CFR_wheel+ angularFR)));
1181    ex Vcreepphi_FR=(sys.Angular_Velocity("xyz1","B_WhsF") * u_nR_FR)
          /((numeric(1,2)*( V_abs_CFR_wheel+angularFR)));
1182
1183 */
1184    Vector3D FKALKER_FR = (Fx_FR*u_t1R_FR) + (Fy_FR * u_t2R_FR);
1185        Vector3D MKALKER_FR = Mz_FR * u_nR_FR;
1186
1187    Drawing3D * D_FKALKER_FR = sys.new_Drawing3D("D_FKALKER_FR",&
          FKALKER_FR, J_FR_Rail ,0 ,1 ,0 ,0.5);
1188
1189        Wrench3D * FKalkerA_FR = sys.new_Wrench3D ( "FKalkerA_FR",
          FKALKER_FR, MKALKER_FR , J_FR_Wheel , WhsF , "Constitutive") ;
1190          Wrench3D * FKalkerR_FR = sys.new_Wrench3D ( "FKalkerR_FR",-
          FKALKER_FR,- MKALKER_FR ,J_FR_Rail , RailF , "Constitutive") ;
```

```
1191
1192        Phi(15,0) = u_t1R_FR * (rW_FR-rR_FR);//u_t1R_FR * (rW_FL-
      rR_FR);Phi(15,0)
1193        Phi(16,0) = u_t2R_FR * (rW_FR-rR_FR);//u_t2R_FR * (rW_FL-
      rR_FR);Phi(16,0)
1194        Phi(17,0) = u_nR_FR  * (rW_FR-rR_FR);//u_nR_FR  * (rW_FL-
      rR_FR);Phi(17,0)
1195        Phi(18,0) = u_nR_FR*t1W_FR;//nR_FR*t1W_FR;Phi(18,0)
1196    Phi(19,0) = u_nR_FR*t2W_FR;//nR_FR*t2W_FR;Phi(19,0)
1197
1198  /*
1199    Phi(5,0) = u_t1R_FR * (rW_FR-rR_FR);//u_t1R_FR * (rW_FL-rR_FR);
      Phi(15,0)
1200        Phi(6,0) = u_t2R_FR * (rW_FR-rR_FR);//u_t2R_FR * (rW_FL-rR_FR
      );Phi(16,0)
1201        Phi(7,0) = u_nR_FR  * (rW_FR-rR_FR);//u_nR_FR  * (rW_FL-rR_FR
      );Phi(17,0)
1202        Phi(8,0) = u_nR_FR*t1W_FR;//nR_FR*t1W_FR;Phi(18,0)
1203        Phi(9,0) = u_nR_FR*t2W_FR;//nR_FR*t2W_FR;Phi(19,0)
1204    */
1205
1206      symbol_numeric lambda16  = *sys.new_Joint_Unknown("lambda16");
1207                    symbol_numeric lambda17  = *sys.new_Joint_Unknown("
      lambda17");
1208          symbol_numeric lambda18  = *sys.new_Joint_Unknown("lambda18
      ");
1209                symbol_numeric lambda19  = *sys.new_Joint_Unknown("
      lambda19");
1210      symbol_numeric lambda20  = *sys.new_Joint_Unknown("lambda20");
1211
1212
1213
1214    Vector3D P_O_WhsB = sys.Position_Vector("O","O_WhsB");
1215    Vector3D P_O_WhsF = sys.Position_Vector("O","O_WhsF");
1216
1217          //
1218  /*
1219          Drawing3D * D_O_RAIL_RL = sys.new_Drawing3D("D_O_RAIL_LR",
      O_RAIL_RL,0.06);
1220          Drawing3D * D_O_RAIL_RR = sys.new_Drawing3D("D_O_RAIL_RR",
      O_RAIL_RR,0.06);
1221          Drawing3D * D_O_RailFL = sys.new_Drawing3D("D_O_RAIL_LF",
      O_RailFL,0.06);
1222          Drawing3D * D_O_RailFR  = sys.new_Drawing3D("D_O_RAIL_RF",
      O_RailFR  ,0.06);
1223
1224          Drawing3D * D_O_WHS_RL = sys.new_Drawing3D("D_O_WHS_LR",
      O_WHS_RL,0.06);
1225          Drawing3D * D_O_WHS_RR = sys.new_Drawing3D("D_O_WHS_RR",
      O_WHS_RR,0.06);
1226          Drawing3D * D_O_WhsFL = sys.new_Drawing3D("D_O_WHS_LF",
      O_WhsFL,0.06);
1227          Drawing3D * D_O_WhsFR = sys.new_Drawing3D("D_O_WHS_RF",
      O_WhsFR,0.06);
1228
1229    Vector3D O_RAIL_RL_JRAIL_RL = *sys.new_Vector3D("
      O_RAIL_RL_JRAIL_RL",0,0,RRAIL,"B_RAILR");
```

```
1230    Vector3D O_RAIL_RR_JRAIL_RR = *sys.new_Vector3D("
          O_RAIL_RR_JRAIL_RR",0,0,RRAIL,"B_RAILR");
1231    Vector3D O_RailFL_JRAIL_FL = *sys.new_Vector3D("O_RailFL_JRAIL_FL
          ",0,0,RRAIL,"B_RailF");
1232    Vector3D O_RailFR _JRAIL_FR = *sys.new_Vector3D("O_RailFR
          _JRAIL_FR",0,0,RRAIL,"B_RailF");
1233
1234        Point * JRAIL_RL = sys.new_Point("JRAIL_RL","O_RAIL_RL",&
          O_RAIL_RL_JRAIL_RL);
1235        Point * JRAIL_RR = sys.new_Point("JRAIL_RR","O_RAIL_RR",&
          O_RAIL_RR_JRAIL_RR);
1236        Point * JRAIL_FL = sys.new_Point("JRAIL_FL","O_RailFL",&
          O_RailFL_JRAIL_FL);
1237        Point * JRAIL_FR = sys.new_Point("JRAIL_FR","O_RailFR ",&
          O_RailFR _JRAIL_FR);
1238
1239    Vector3D O_WHS_RL_JWHS_RL = *sys.new_Vector3D("O_WHS_RL_JWHS_RL
          ",0,0,RWHS,"B_WHSR");
1240    Vector3D O_WHS_RR_JWHS_RR = *sys.new_Vector3D("O_WHS_RR_JWHS_RR
          ",0,0,RWHS,"B_WHSR");
1241    Vector3D O_WhsFL_JWHS_FL = *sys.new_Vector3D("O_WhsFL_JWHS_FL
          ",0,0,RWHS,"B_WhsF");
1242    Vector3D O_WhsFR_JWHS_FR = *sys.new_Vector3D("O_WhsFR_JWHS_FR
          ",0,0,RWHS,"B_WhsF");
1243
1244        Point * JWHS_RL = sys.new_Point("JWHS_RL","O_RAIL_RL",&
          O_WHS_RL_JWHS_RL);
1245        Point * JWHS_RR = sys.new_Point("JWHS_RR","O_RAIL_RR",&
          O_WHS_RR_JWHS_RR);
1246        Point * JWHS_FL = sys.new_Point("JWHS_FL","O_RailFL",&
          O_WhsFL_JWHS_FL);
1247        Point * JWHS_FR = sys.new_Point("JWHS_FR","O_RailFR ",&
          O_WhsFR_JWHS_FR);
1248
1249        Drawing3D * D_JRAIL_RL = sys.new_Drawing3D("D_JRAIL_RL",
          JRAIL_RL,0.03);
1250        Drawing3D * D_JRAIL_RR = sys.new_Drawing3D("D_JRAIL_RR",
          JRAIL_RR,0.03);
1251        Drawing3D * D_JRAIL_FL = sys.new_Drawing3D("D_JRAIL_FL",
          JRAIL_FL,0.03);
1252        Drawing3D * D_JRAIL_FR = sys.new_Drawing3D("D_JRAIL_FR",
          JRAIL_FR,0.03);
1253
1254        Drawing3D * D_JWHS_RL = sys.new_Drawing3D("D_JWHS_RL",
          JWHS_RL,0.03);
1255        Drawing3D * D_JWHS_RR = sys.new_Drawing3D("D_JWHS_RR",
          JWHS_RR,0.03);
1256        Drawing3D * D_JWHS_FL = sys.new_Drawing3D("D_JWHS_FL",
          JWHS_FL,0.03);
1257        Drawing3D * D_JWHS_FR = sys.new_Drawing3D("D_JWHS_FR",
          JWHS_FR,0.03);
1258
1259 */
1260
1261
1262
```

```
1263 //
     ********************************************************************************
1264 //  Radius of the Ellipse
1265 //
     ********************************************************************************
1266
1267   cout << "Radius of the Ellipse" << endl;
1268
1269    //BACK LEFT
1270     ex cryW_BL = abs(sqrt(fW_BL(0,0)*fW_BL(0,0) + fW_BL(2,0)*fW_BL
       (2,0)));
1271       ex crxW_BL = abs(pow(1 + pow(sys.diff(splineW_BL,uW_BL),2.0)
       ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineW_BL,uW_BL),uW_BL));
         /* Radius of Curvature rho = (1+y')^(3/2) / y'' */
1272     ex cryR_BL = abs(sqrt(fR_BL(0,0)*fR_BL(0,0) + fR_BL(2,0)*fR_BL
       (2,0)));
1273     ex crxR_BL = abs(pow(1 + pow(sys.diff(splineR_BL,uR_BL),2.0)
       ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineR_BL,uR_BL),uR_BL));
1274
1275
1276   ex n_BL = sqrt(lambda1*lambda1+lambda2*lambda2+lambda3*lambda3);
1277
1278   Matrix Radius_BL(10,1);
1279
1280    Radius_BL(0,0)=cryW_BL;
1281    Radius_BL(1,0)=crxW_BL;
1282    Radius_BL(2,0)=cryR_BL;
1283    Radius_BL(3,0)=crxR_BL;
1284    Radius_BL(4,0)=n_BL;
1285    Radius_BL(5,0)=Vcreepx_BL;
1286    Radius_BL(6,0)=Vcreepy_BL;
1287    Radius_BL(7,0)=Vcreepphi_BL;
1288    Radius_BL(8,0)=Vprom_BL_W_Rail;
1289    Radius_BL(9,0)=Vprom_BL_W_Rail2;
1290 lst new_atom_list_Radius_BL, new_exp_list_Radius_BL;
1291     matrix_list_optimize ( Radius_BL, new_atom_list_Radius_BL,
       new_exp_list_Radius_BL);
1292
1293     sys.export_function_MATLAB("Radius_BL", "Radius_BL_",
       Radius_BL, new_atom_list_Radius_BL, new_exp_list_Radius_BL);
1294
1295
1296
1297
1298
1299
1300   //BACK RIGHT
1301     ex cryW_BR = abs(sqrt(fW_BR(0,0)*fW_BR(0,0) + fW_BR(2,0)*fW_BR
       (2,0)));
1302       ex crxW_BR = abs(pow(1 + pow(sys.diff(splineW_BR,uW_BR),2.0)
       ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineW_BR,uW_BR),uW_BR));
         /* Radius of Curvature rho = (1+y')^(3/2) / y'' */
1303     ex cryR_BR = abs(sqrt(fR_BR(0,0)*fR_BR(0,0) + fR_BR(2,0)*fR_BR
       (2,0)));
```

```
1304      ex crxR_BR = abs(pow(1 + pow(sys.diff(splineR_BR,uR_BR),2.0)
          ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineR_BR,uR_BR),uR_BR));
1305
1306
1307    ex n_BR = sqrt(lambda6*lambda6+lambda7*lambda7+lambda8*lambda8);
1308
1309    Matrix Radius_BR(10,1);
1310
1311     Radius_BR(0,0)=cryW_BR;
1312     Radius_BR(1,0)=crxW_BR;
1313     Radius_BR(2,0)=cryR_BR;
1314     Radius_BR(3,0)=crxR_BR;
1315     Radius_BR(4,0)=n_BR;
1316     Radius_BR(5,0)=Vcreepx_BR;
1317     Radius_BR(6,0)=Vcreepy_BR;
1318     Radius_BR(7,0)=Vcreepphi_BR;
1319     Radius_BR(8,0)=Vprom_BR_W_Rail;
1320     Radius_BR(9,0)=Vprom_BR_W_Rail2;
1321
1322  lst new_atom_list_Radius_BR, new_exp_list_Radius_BR;
1323      matrix_list_optimize ( Radius_BR, new_atom_list_Radius_BR,
          new_exp_list_Radius_BR);
1324
1325      sys.export_function_MATLAB("Radius_BR", "Radius_BR_",
          Radius_BR, new_atom_list_Radius_BR, new_exp_list_Radius_BR);
1326
1327    //FRONT LEFT
1328      ex cryW_FL = abs(sqrt(fW_FL(0,0)*fW_FL(0,0) + fW_FL(2,0)*fW_FL
          (2,0)));
1329        ex crxW_FL = abs(pow(1 + pow(sys.diff(splineW_FL,uW_FL),2.0)
          ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineW_FL,uW_FL),uW_FL));
          /* Radius of Curvature rho = (1+y')^(3/2) / y'' */
1330      ex cryR_FL = abs(sqrt(fR_FL(0,0)*fR_FL(0,0) + fR_FL(2,0)*fR_FL
          (2,0)));
1331      ex crxR_FL = abs(pow(1 + pow(sys.diff(splineR_FL,uR_FL),2.0)
          ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineR_FL,uR_FL),uR_FL));
1332
1333
1334    ex n_FL = sqrt(lambda11*lambda11+lambda12*lambda12+lambda13*
          lambda13);
1335
1336      Matrix Radius_FL(10,1);
1337
1338     Radius_FL(0,0)=cryW_FL;
1339     Radius_FL(1,0)=crxW_FL;
1340     Radius_FL(2,0)=cryR_FL;
1341     Radius_FL(3,0)=crxR_FL;
1342     Radius_FL(4,0)=n_FL;
1343     Radius_FL(5,0)=Vcreepx_FL;
1344     Radius_FL(6,0)=Vcreepy_FL;
1345     Radius_FL(7,0)=Vcreepphi_FL;
1346     Radius_FL(8,0)=Vprom_FL_W_Rail;
1347     Radius_FL(9,0)=Vprom_FL_W_Rail2;
1348
1349  lst new_atom_list_Radius_FL, new_exp_list_Radius_FL;
1350      matrix_list_optimize ( Radius_FL, new_atom_list_Radius_FL,
          new_exp_list_Radius_FL);
```

```
1351
1352      sys.export_function_MATLAB("Radius_FL", "Radius_FL_",
          Radius_FL, new_atom_list_Radius_FL, new_exp_list_Radius_FL);
1353
1354    //FRONT RIGHT
1355
1356      ex cryW_FR = abs(sqrt(fW_FR(0,0)*fW_FR(0,0) + fW_FR(2,0)*fW_FR
          (2,0)));
1357        ex crxW_FR = abs(pow(1 + pow(sys.diff(splineW_FR,uW_FR),2.0)
          ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineW_FR,uW_FR),uW_FR));
          /* Radius of Curvature rho = (1+y')^(3/2) / y'' */
1358      ex cryR_FR = abs(sqrt(fR_FR(0,0)*fR_FR(0,0) + fR_FR(2,0)*fR_FR
          (2,0)));
1359      ex crxR_FR = abs(pow(1 + pow(sys.diff(splineR_FR,uR_FR),2.0)
          ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineR_FR,uR_FR),uR_FR));
1360
1361
1362    ex n_FR = sqrt(lambda16*lambda16+lambda17*lambda17+lambda18*
          lambda18);
1363
1364     Matrix Radius_FR(10,1);
1365
1366     Radius_FR(0,0)=cryW_FR;
1367     Radius_FR(1,0)=crxW_FR;
1368     Radius_FR(2,0)=cryR_FR;
1369     Radius_FR(3,0)=crxR_FR;
1370     Radius_FR(4,0)=n_FR;
1371     Radius_FR(5,0)=Vcreepx_FR;
1372     Radius_FR(6,0)=Vcreepy_FR;
1373     Radius_FR(7,0)=Vcreepphi_FR;
1374     Radius_FR(8,0)=Vprom_FR_W_Rail;
1375     Radius_FR(9,0)=Vprom_FR_W_Rail2;
1376
1377  lst new_atom_list_Radius_FR, new_exp_list_Radius_FR;
1378      matrix_list_optimize ( Radius_FR, new_atom_list_Radius_FR,
          new_exp_list_Radius_FR);
1379
1380      sys.export_function_MATLAB("Radius_FR", "Radius_FR_",
          Radius_FR, new_atom_list_Radius_FR, new_exp_list_Radius_FR);
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393  //
          ************************************************************************************
1394  //  Glide forces
1395  //
          ************************************************************************************
```

```
1396
1397   cout << "Glide forces" << endl;
1398
1399     symbol_numeric  Ct = *sys.new_Parameter("Ct",10000.000);
1400
1401     Vector3D F1_BR = -Ct * Vcon_BR_W_Rail;
1402     Vector3D F1_BL = -Ct * Vcon_BL_W_Rail;
1403     Vector3D F1_FL = -Ct * Vcon_FL_W_Rail;
1404     Vector3D F1_FR = -Ct * Vcon_FR_W_Rail;
1405
1406
1407     cout << F1_BR << endl;
1408     cout << F1_BL << endl;
1409     cout << F1_FL << endl;
1410     cout << F1_FR << endl;
1411     cout << F1_FR(0,0) << endl;
1412
1413     Matrix Kalker_Kinematics(12,1);
1414
1415     Kalker_Kinematics(0,0)=F1_BR*u_t1R_BR;//Fx_BR
1416     Kalker_Kinematics(1,0)=F1_BR*u_t2R_BR;//Fy_BR
1417     Kalker_Kinematics(2,0)=0;//Mz_BR
1418     Kalker_Kinematics(3,0)=F1_BL*u_t1R_BL;//Fx_BL
1419     Kalker_Kinematics(4,0)=F1_BL*u_t2R_BL;//Fy_bL
1420     Kalker_Kinematics(5,0)=0;//Mz_BL
1421     Kalker_Kinematics(6,0)=F1_FL*u_t1R_FL;//Fx_FL
1422     Kalker_Kinematics(7,0)=F1_FL*u_t2R_FL;//Fy_FL
1423     Kalker_Kinematics(8,0)=0;//Mz_FL
1424     Kalker_Kinematics(9,0)=F1_FR*u_t1R_FR;//Fx_FR
1425     Kalker_Kinematics(10,0)=F1_FR*u_t2R_FR;//Fy_FR
1426     Kalker_Kinematics(11,0)=0;//Mz_FR
1427
1428
1429     lst new_atom_list_Kalker_Kinematics,
        new_exp_list_Kalker_Kinematics;
1430     matrix_list_optimize (Kalker_Kinematics,
        new_atom_list_Kalker_Kinematics, new_exp_list_Kalker_Kinematics
        );
1431
1432     sys.export_function_MATLAB("Kalker_Kinematics", "
        Kalker_Kinematics_", Kalker_Kinematics,
        new_atom_list_Kalker_Kinematics, new_exp_list_Kalker_Kinematics
         );
1433
1434
1435 //
        ********************************************************************************
1436 //  Vectors needed to take diferent Jacobians
1437 //
        ********************************************************************************
1438
1439   cout << "Vectors needed to take diferent Jacobians" << endl;
1440
1441   Matrix q = sys.Coordinates();
```

```
1442    Matrix dq = sys.Velocities();
1443    Matrix ddq = sys.Accelerations();
1444    Matrix epsilon = sys.Joint_Unknowns();
1445    Matrix pars = sys.Parameters();
1446    Matrix inps = sys.Inputs();
1447    Matrix unks = sys.Joint_Unknowns();
1448
1449  //
        ********************************************************************************************
1450  //  Dynamic Equations
1451  //
        ********************************************************************************************
1452
1453    cout << "Dynamic Equations" << endl;
1454
1455    Matrix Dynamic_Equations = sys.GenForceSys("ALL");
1456
1457
1458  //
        ********************************************************************************************
1459  //  Output Vector
1460  //
        ********************************************************************************************
1461
1462    cout << "Output Vector" << endl;
1463
1464    Matrix Output(36,1);//Matrix Output(40,1);
1465    Output(0,0)=u_nR_BL(0,0);
1466    Output(1,0)=u_nR_BL(1,0);
1467    Output(2,0)=u_nR_BL(2,0);
1468    Output(3,0)= u_nR_BR(0,0);
1469    Output(4,0)= u_nR_BR(1,0);
1470    Output(5,0)= u_nR_BR(2,0);
1471    Output(6,0)= u_nR_FL(0,0);
1472    Output(7,0)= u_nR_FL(1,0);
1473    Output(8,0)= u_nR_FL(2,0);
1474    Output(9,0)= u_nR_FR(0,0);
1475    Output(10,0)= u_nR_FR(1,0);
1476    Output(11,0)= u_nR_FR(2,0);
1477
1478    Output(12,0)=u_t1R_BL(0,0);
1479    Output(13,0)=u_t1R_BL(1,0);
1480    Output(14,0)=u_t1R_BL(2,0);
1481    Output(15,0)= u_t1R_BR(0,0);
1482    Output(16,0)= u_t1R_BR(1,0);
1483    Output(17,0)= u_t1R_BR(2,0);
1484    Output(18,0)= u_t1R_FL(0,0);
1485    Output(19,0)= u_t1R_FL(1,0);
1486    Output(20,0)= u_t1R_FL(2,0);
1487    Output(21,0)= u_t1R_FR(0,0);
1488    Output(22,0)= u_t1R_FR(1,0);
1489    Output(23,0)= u_t1R_FR(2,0);
1490
```

```
1491
1492    Output(24,0)=u_t2R_BL(0,0);
1493    Output(25,0)=u_t2R_BL(1,0);
1494    Output(26,0)=u_t2R_BL(2,0);
1495    Output(27,0)= u_t2R_BR(0,0);
1496    Output(28,0)= u_t2R_BR(1,0);
1497    Output(29,0)= u_t2R_BR(2,0);
1498    Output(30,0)= u_t2R_FL(0,0);
1499    Output(31,0)= u_t2R_FL(1,0);
1500    Output(32,0)= u_t2R_FL(2,0);
1501
1502    Output(33,0)= u_t2R_FR(0,0);
1503    Output(34,0)= u_t2R_FR(1,0);
1504    Output(35,0)= u_t2R_FR(2,0);
1505
1506
1507
1508
1509   */
1510      //REAR LEFT
1511        ex cryW_RL =abs(sqrt(fW_RL(0,0)*fW_RL(0,0) + fW_RL(2,0)*fW_RL
       (2,0)));
1512        ex crxW_RL = abs(pow(1 + pow(sys.diff(splineW_RL,uW_RL),2.0)
       ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineW_RL,uW_RL),uW_RL));
          /* Radius of Curvature rho = (1+y')^(3/2) / y'' */
1513
1514  /*
1515
1516      ex cryR_RL = abs(sqrt(fW_RL(0,0)*fW_RL(0,0) + fW_RL(2,0)*fW_RL
       (2,0)));
1517      ex crxR_RL = abs(pow(1 + pow(sys.diff(splineR_RL,uR_RL),2.0)
       ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineR_RL,uR_RL),uR_RL));
1518
1519        Output(0,0) = sqrt(lambda1*lambda1+lambda2*lambda2+lambda3*
       lambda3);//0.0;
1520        Output(1,0) = (*dbRAILR * cryR_RL - *dbWHSR * cryW_RL); //(*
       dbRAIL * cryR_RLR - *dbRWhsF * cryW_RLR);
1521        Output(2,0) = (*daCh * cryR_RL - *dbWHSR * cryW_RL);//(*daCh
       * cryR_RLR - *daRWhsF * cryW_RLR);
1522        Output(3,0) = (sys.Angular_Velocity("xyz1","B_WHSR") *
       u_nR_RL);
1523        Output(4,0) = *dbWHSR;
1524        Output(5,0) = *dxCh;
1525        Output(6,0) = cryW_RL;
1526        Output(7,0) = crxW_RL;
1527        Output(8,0) = cryR_RL;
1528        Output(9,0) = crxR_RL;
1529
1530      //REAR RIGHT
1531        ex cryW_RR = abs(sqrt(fW_RR(0,0)*fW_RR(0,0) + fW_RR(2,0)*
       fW_RR(2,0)));
1532        ex crxW_RR = abs(pow(1 + pow(sys.diff(splineW_RR,uW_RR),2.0)
       ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineW_RR,uW_RR),uW_RR));
          /* Radius of Curvature rho = (1+y')^(3/2) / y'' */
1533   /*     ex cryR_RR = abs(sqrt(fW_RR(0,0)*fW_RR(0,0) + fW_RR(2,0)*
       fW_RR(2,0)));
```

```
1534    ex crxR_RR = abs(pow(1 + pow(sys.diff(splineR_RR,uR_RR),2.0)
        ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineR_RR,uR_RR),uR_RR));
1535
1536    Output(10,0) = sqrt(lambda6*lambda6+lambda7*lambda7+lambda8*
        lambda8);
1537    Output(11,0) = (*dbRAILR * cryR_RR - *dbWHSR * cryW_RR);//(*
        dbRAIL * cryR_RFR - *dbRWhsF * cryW_RFR);
1538    Output(12,0) = (*daCh * cryR_RR- *dbWHSR * cryW_RR);//(*daCh
        * cryR_RFR - *daRWhsF * cryW_RFR);
1539    Output(13,0) = (sys.Angular_Velocity("xyz1","B_WHSR") *
        u_nR_RR);
1540    Output(14,0) = *dbWHSR;
1541    Output(15,0) = *dxCh;
1542    Output(16,0) = cryW_RR;
1543    Output(17,0) = crxW_RR;
1544    Output(18,0) = cryR_RR;
1545    Output(19,0) = crxR_RR;
1546
1547  //FRONT LEFT
1548    ex cryW_FL = abs(sqrt(fW_FL(0,0)*fW_FL(0,0) + fW_FL(2,0)*
        fW_FL(2,0)));
1549    ex crxW_FL = abs(pow(1 + pow(sys.diff(splineW_FL,uW_FL),2.0)
        ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineW_FL,uW_FL),uW_FL));
      /* Radius of Curvature rho = (1+y')^(3/2) / y'' */
1550  /*  ex cryR_FL = abs(sqrt(fW_FL(0,0)*fW_FL(0,0) + fW_FL(2,0)*
        fW_FL(2,0)));
1551    ex crxR_FL = abs(pow(1 + pow(sys.diff(splineR_FL,uR_FL),2.0)
        ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineR_FL,uR_FL),uR_FL));
1552
1553    Output(20,0) = sqrt(lambda11*lambda11+lambda12*lambda12+
        lambda13*lambda13);
1554    Output(21,0) = (*dbRailF * cryR_FL - *dbWhsF * cryW_FL);//(*
        dbRAIL * cryR_RFR - *dbRWhsF * cryW_RFR);
1555    Output(22,0) = (*daCh *  cryR_FL- *dbWhsF * cryW_FL);//(*daCh
         * cryR_RFR - *daRWhsF * cryW_RFR);
1556    Output(23,0) = (sys.Angular_Velocity("xyz1","B_WhsF") *
        u_nR_FL);
1557    Output(24,0) = *dbWhsF;
1558    Output(25,0) = *dxCh;
1559    Output(26,0) = cryW_FL;
1560    Output(27,0) = crxW_FL;
1561    Output(28,0) = cryR_FL;
1562    Output(29,0) = crxR_FL;
1563
1564  //FRONT RIGHT
1565    ex cryW_FR = abs(sqrt(fW_FR(0,0)*fW_FR(0,0) + fW_FR(2,0)*
        fW_FR(2,0)));
1566    ex crxW_FR = abs(pow(1 + pow(sys.diff(splineW_FR,uW_FR),2.0)
        ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineW_FR,uW_FR),uW_FR));
      /* Radius of Curvature rho = (1+y')^(3/2) / y'' */
1567  /*  ex cryR_FR = abs(sqrt(fW_FR(0,0)*fW_FR(0,0) + fW_FR(2,0)*
        fW_FR(2,0)));
1568    ex crxR_FR = abs(pow(1 + pow(sys.diff(splineR_FR,uR_FR),2.0)
        ,(3.0/2.0))) / abs(sys.diff(sys.diff(splineR_FR,uR_FR),uR_FR));
1569
1570    Output(30,0) = sqrt(lambda16*lambda16+lambda17*lambda17+
        lambda18*lambda18);
```

```
1571        Output(31,0) = (*dbRailF * crxR_FR - *dbWhsF * cryW_FR);//(*
    dbRailF * cryR_RFR - *dbRWhsF * cryW_RFR);
1572        Output(32,0) = (*dbRailF * crxR_FR- *dbWhsF * cryW_FR);//(*
    daCh * cryR_RFR - *daRWhsF * cryW_RFR);
1573        Output(33,0) = (sys.Angular_Velocity("xyz1","B_WhsF") *
    u_nR_FR);
1574        Output(34,0) = *dbWhsF;
1575        Output(35,0) = *dxCh;
1576        Output(36,0) = cryW_FR;
1577        Output(37,0) = crxW_FR;
1578        Output(38,0) = cryR_FR;
1579        Output(39,0) = crxR_FR;
1580  */
1581    sys.new_Matrix("Output",Output);
1582
1583  //
      ***************************************************************************************
1584  //  Energy Equations
1585  //
      ***************************************************************************************
1586
1587    cout << "Energy Equations" << endl;
1588
1589    Matrix Energy(1,1);
1590    Energy(0,0)=0;
1591    sys.new_Matrix("Energy",Energy);
1592
1593  //
      ***************************************************************************************
1594  //  KALKER
1595  //
      ***************************************************************************************
1596
1597        // KALKER
1598
1599      symbol_numeric E_elastic = *sys.new_Parameter("E_elastic",210.0
      e+9);  // MPa
1600      symbol_numeric nu_poisson = *sys.new_Parameter("nu_poisson"
      ,0.27);      // Steel
1601
1602          // Steel
1603
1604      symbol_numeric G_elastic = *sys.new_Parameter("G_elastic",80.7e
      +9);    // G = E/(2*(1+nu)); %MPa
1605
1606        // hertz ellipse parameters
1607      symbol_numeric aBR = *sys.new_Parameter("aBR",0.0);
1608      symbol_numeric bBR = *sys.new_Parameter("bBR",0.0);
1609      symbol_numeric aBL = *sys.new_Parameter("aBL",0.0);
1610      symbol_numeric bBL = *sys.new_Parameter("bBL",0.0);
1611
1612      symbol_numeric aFR = *sys.new_Parameter("aFR",0.0);
1613      symbol_numeric bFR = *sys.new_Parameter("bFR",0.0);
```

```
1614      symbol_numeric aFL = *sys.new_Parameter("aFL",0.0);
1615      symbol_numeric bFL = *sys.new_Parameter("bFL",0.0);
1616
1617
1618    // Kalker's coefficients
1619
1620      symbol_numeric C11BR = *sys.new_Parameter("C11BR",0.0);
1621      symbol_numeric C22BR = *sys.new_Parameter("C22BR",0.0);
1622      symbol_numeric C23BR = *sys.new_Parameter("C23BR",0.0);
1623      symbol_numeric C33BR = *sys.new_Parameter("C33BR",0.0);
1624
1625      symbol_numeric C11BL = *sys.new_Parameter("C11BL",0.0);
1626      symbol_numeric C22BL = *sys.new_Parameter("C22BL",0.0);
1627      symbol_numeric C23BL = *sys.new_Parameter("C23BL",0.0);
1628      symbol_numeric C33BL = *sys.new_Parameter("C33BL",0.0);
1629
1630      symbol_numeric C11FR = *sys.new_Parameter("C11FR",0.0);
1631      symbol_numeric C22FR = *sys.new_Parameter("C22FR",0.0);
1632      symbol_numeric C23FR = *sys.new_Parameter("C23FR",0.0);
1633      symbol_numeric C33FR = *sys.new_Parameter("C33FR",0.0);
1634
1635      symbol_numeric C11FL = *sys.new_Parameter("C11FL",0.0);
1636      symbol_numeric C22FL = *sys.new_Parameter("C22FL",0.0);
1637      symbol_numeric C23FL = *sys.new_Parameter("C23FL",0.0);
1638      symbol_numeric C33FL = *sys.new_Parameter("C33FL",0.0);
1639
1640
1641    Matrix D_BR(3,3);
1642      ex GabBR = G_elastic*aBR*bBR;
1643      D_BR(0,0) = GabBR * C11BR; D_BR(0,1) = 0;
          D_BR(0,2) = 0;
1644      D_BR(1,0) = 0;             D_BR(1,1) = GabBR * C22BR;
          D_BR(1,2) = GabBR * sqrt(aBR*bBR)*C23BR;
1645      D_BR(2,0) = 0;             D_BR(2,1) =-GabBR * sqrt(aBR*bBR)*
          C23BR;   D_BR(2,2) = GabBR * aBR*bBR*C33BR;
1646
1647      Matrix D_BL(3,3);
1648      ex GabBL = G_elastic*aBL*bBL;
1649      D_BL(0,0) = GabBL * C11BL; D_BL(0,1) = 0;
          D_BL(0,2) = 0;
1650      D_BL(1,0) = 0;             D_BL(1,1) = GabBL * C22BL;
          D_BL(1,2) = GabBL * sqrt(aBL*bBL)*C23BL;
1651      D_BL(2,0) = 0;             D_BL(2,1) =-GabBL * sqrt(aBL*bBL)*
          C23BL;   D_BL(2,2) = GabBL * aBL*bBL*C33BL;
1652
1653
1654
1655      Matrix D_FR(3,3);
1656      ex GabFR = G_elastic*aFR*bFR;
1657      D_FR(0,0) = GabFR * C11FR; D_FR(0,1) = 0;
          D_FR(0,2) = 0;
1658      D_FR(1,0) = 0;             D_FR(1,1) = GabFR * C22FR;
          D_FR(1,2) = GabFR * sqrt(aFR*bFR)*C23FR;
1659      D_FR(2,0) = 0;             D_FR(2,1) = -GabFR * sqrt(aFR*bFR)*
          C23FR;   D_FR(2,2) = GabFR * aFR*bFR*C33FR;
1660
1661      Matrix D_FL(3,3);
```

```
1662      ex GabFL = G_elastic*aFL*bFL;
1663      D_FL(0,0) = GabFL * C11FL; D_FL(0,1) = 0;
            D_FL(0,2) = 0;
1664      D_FL(1,0) = 0;            D_FL(1,1) = GabFL * C22FL;
            D_FL(1,2) = GabFL * sqrt(aFL*bFL)*C23FL;
1665      D_FL(2,0) = 0;            D_FL(2,1) = -GabFL * sqrt(aFL*bFL)*
          C23FL;   D_FL(2,2) = GabFL * aFL*bFL*C33FL;
1666  //
          ********************************************************************************************

1667  //   Matrix Calculation
1668  //
          ********************************************************************************************

1669
1670          cout << "Matrix Calculation" << endl;
1671
1672         //sys.Matrix_Calculation(Phi, coord_indep_init,
          vel_indep_init,  sys, METHOD);
1673    sys.Matrix_Calculation(Phi,coord_indep_init , vel_indep_init ,
          Dynamic_Equations, sys, METHOD);
1674
1675
1676
1677  //
          ********************************************************************************************

1678  //   Kalker NEW
1679  //
          ********************************************************************************************

1680      Matrix Qo   = *sys.get_Matrix( "Q" );
1681
1682      lst kalker_forcesBR, kalker_forcesBL,kalker_forcesFR,
          kalker_forcesFL;
1683    kalker_forcesBR =Fx_BR,Fy_BR,Mz_BR;
1684    kalker_forcesBL =Fx_BL,Fy_BL,Mz_BL;
1685    kalker_forcesFR =Fx_FR,Fy_FR,Mz_FR;
1686    kalker_forcesFL =Fx_FL,Fy_FL,Mz_FL;
1687
1688      Matrix Mat_KakerForcesBR = Matrix(kalker_forcesBR.nops(),1,
          kalker_forcesBR);
1689      Matrix Mat_KakerForcesBL = Matrix(kalker_forcesBL.nops(),1,
          kalker_forcesBL);
1690      Matrix Mat_KakerForcesFR = Matrix(kalker_forcesFR.nops(),1,
          kalker_forcesFR);
1691      Matrix Mat_KakerForcesFL = Matrix(kalker_forcesFL.nops(),1,
          kalker_forcesFL);
1692
1693           lst new_atom_list_Mat_KakerForcesBR,
          new_exp_list_Mat_KakerForcesBR;
1694      matrix_list_optimize (Mat_KakerForcesBR,
          new_atom_list_Mat_KakerForcesBR, new_exp_list_Mat_KakerForcesBR
          );
1695      sys.export_Matrix_C ("Mat_KakerForcesBR","_Mat_KakerForcesBR",
          Mat_KakerForcesBR, new_atom_list_Mat_KakerForcesBR,
          new_exp_list_Mat_KakerForcesBR ,ORDER);
```

```
1696    sys.export_function_MATLAB("Mat_KakerForcesBR", "
        Mat_KakerForcesBR_",Mat_KakerForcesBR,
        new_atom_list_Mat_KakerForcesBR, new_exp_list_Mat_KakerForcesBR
        );
1697
1698        lst new_atom_list_Mat_KakerForcesBL,
        new_exp_list_Mat_KakerForcesBL;
1699    matrix_list_optimize (Mat_KakerForcesBL,
        new_atom_list_Mat_KakerForcesBL, new_exp_list_Mat_KakerForcesBL
        );
1700    sys.export_Matrix_C ("Mat_KakerForcesBL","_Mat_KakerForcesBL",
        Mat_KakerForcesBL, new_atom_list_Mat_KakerForcesBL,
        new_exp_list_Mat_KakerForcesBL ,ORDER);
1701    sys.export_function_MATLAB("Mat_KakerForcesBL", "
        Mat_KakerForcesBL_",Mat_KakerForcesBL,
        new_atom_list_Mat_KakerForcesBL, new_exp_list_Mat_KakerForcesBL
        );
1702
1703        lst new_atom_list_Mat_KakerForcesFR,
        new_exp_list_Mat_KakerForcesFR;
1704    matrix_list_optimize (Mat_KakerForcesFR,
        new_atom_list_Mat_KakerForcesFR, new_exp_list_Mat_KakerForcesFR
        );
1705    sys.export_Matrix_C ("Mat_KakerForcesFR","_Mat_KakerForcesFR",
        Mat_KakerForcesFR, new_atom_list_Mat_KakerForcesFR,
        new_exp_list_Mat_KakerForcesFR ,ORDER);
1706    sys.export_function_MATLAB("Mat_KakerForcesFR", "
        Mat_KakerForcesFR_",Mat_KakerForcesFR,
        new_atom_list_Mat_KakerForcesFR, new_exp_list_Mat_KakerForcesFR
        );
1707
1708        lst new_atom_list_Mat_KakerForcesFL,
        new_exp_list_Mat_KakerForcesFL;
1709    matrix_list_optimize (Mat_KakerForcesFL,
        new_atom_list_Mat_KakerForcesFL, new_exp_list_Mat_KakerForcesFL
        );
1710    sys.export_Matrix_C ("Mat_KakerForcesFL","_Mat_KakerForcesFL",
        Mat_KakerForcesFL, new_atom_list_Mat_KakerForcesFL,
        new_exp_list_Mat_KakerForcesFL ,ORDER);
1711    sys.export_function_MATLAB("Mat_KakerForcesFL", "
        Mat_KakerForcesFL_",Mat_KakerForcesFL,
        new_atom_list_Mat_KakerForcesFL, new_exp_list_Mat_KakerForcesFL
        );
1712
1713    Matrix dQ_dKFBR=sys.jacobian(Qo.transpose(),Mat_KakerForcesBR);
1714    Matrix dQ_dKFBL=sys.jacobian(Qo.transpose(),Mat_KakerForcesBL);
1715    Matrix dQ_dKFFR=sys.jacobian(Qo.transpose(),Mat_KakerForcesFR);
1716    Matrix dQ_dKFFL=sys.jacobian(Qo.transpose(),Mat_KakerForcesFL);
1717
1718    lst new_atom_list_dQ_dKFBR, new_exp_list_dQ_dKFBR;
1719    matrix_list_optimize (dQ_dKFBR, new_atom_list_dQ_dKFBR,
        new_exp_list_dQ_dKFBR);
1720    sys.export_Matrix_C ("dQ_dKFBR","_dQ_dKFBR",dQ_dKFBR,
        new_atom_list_dQ_dKFBR, new_exp_list_dQ_dKFBR ,ORDER);
1721    sys.export_function_MATLAB("dQ_dKFBR", "dQ_dKFBR_",dQ_dKFBR,
        new_atom_list_dQ_dKFBR, new_exp_list_dQ_dKFBR);
1722
```

```
1723
1724        lst new_atom_list_dQ_dKFBL, new_exp_list_dQ_dKFBL;
1725        matrix_list_optimize (dQ_dKFBL, new_atom_list_dQ_dKFBL,
            new_exp_list_dQ_dKFBL);
1726        sys.export_Matrix_C ("dQ_dKFBL","_dQ_dKFBL",dQ_dKFBL,
            new_atom_list_dQ_dKFBL, new_exp_list_dQ_dKFBL ,ORDER);
1727        sys.export_function_MATLAB("dQ_dKFBL", "dQ_dKFBL_",dQ_dKFBL,
            new_atom_list_dQ_dKFBL, new_exp_list_dQ_dKFBL);
1728
1729        lst new_atom_list_dQ_dKFFR, new_exp_list_dQ_dKFFR;
1730        matrix_list_optimize (dQ_dKFFR, new_atom_list_dQ_dKFFR,
            new_exp_list_dQ_dKFFR);
1731        sys.export_Matrix_C ("dQ_dKFFR","_dQ_dKFFR",dQ_dKFFR,
            new_atom_list_dQ_dKFFR, new_exp_list_dQ_dKFFR ,ORDER);
1732        sys.export_function_MATLAB("dQ_dKFFR", "dQ_dKFFR_",dQ_dKFFR,
            new_atom_list_dQ_dKFFR, new_exp_list_dQ_dKFFR);
1733
1734
1735        lst new_atom_list_dQ_dKFFL, new_exp_list_dQ_dKFFL;
1736        matrix_list_optimize (dQ_dKFFL, new_atom_list_dQ_dKFFL,
            new_exp_list_dQ_dKFFL);
1737        sys.export_Matrix_C ("dQ_dKFFL","_dQ_dKFFL",dQ_dKFFL,
            new_atom_list_dQ_dKFFL, new_exp_list_dQ_dKFFL ,ORDER);
1738        sys.export_function_MATLAB("dQ_dKFFL", "dQ_dKFFL_",dQ_dKFFL,
            new_atom_list_dQ_dKFFL, new_exp_list_dQ_dKFFL);
1739
1740        Matrix SlipBR (3,1);
1741        SlipBR (0,0) =Vcreepx_BR;
1742        SlipBR (1,0) = Vcreepy_BR;
1743        SlipBR (2,0) = Vcreepphi_BR;
1744
1745        Matrix SlipBL (3,1);
1746        SlipBL (0,0) = Vcreepx_BL;
1747        SlipBL (1,0) = Vcreepy_BL;
1748        SlipBL (2,0) = Vcreepphi_BL;
1749
1750        Matrix SlipFR (3,1);
1751        SlipFR (0,0) =Vcreepx_FR;
1752        SlipFR (1,0) = Vcreepy_FR;
1753        SlipFR (2,0) = Vcreepphi_FR;
1754
1755        Matrix SlipFL (3,1);
1756        SlipFL (0,0) = Vcreepx_FL;
1757        SlipFL (1,0) = Vcreepy_FL;
1758        SlipFL (2,0) = Vcreepphi_FL;
1759
1760
1761        Matrix dSBR_dq = sys.jacobian(SlipBR.transpose(),dq);
1762        Matrix dSBL_dq = sys.jacobian(SlipBL.transpose(),dq);
1763        Matrix dSFR_dq = sys.jacobian(SlipFR.transpose(),dq);
1764        Matrix dSFL_dq = sys.jacobian(SlipFL.transpose(),dq);
1765
1766
1767
1768
1769
```

```
1770        Matrix CKBR = dQ_dKFBR * D_BR*sys.jacobian(SlipBR.transpose(),
            dq);
1771        Matrix CKBL = dQ_dKFBL * D_BL*sys.jacobian(SlipBL.transpose(),
            dq);
1772        Matrix CKFR = dQ_dKFFR * D_FR*sys.jacobian(SlipFR.transpose(),
            dq);
1773        Matrix CKFL = dQ_dKFFL * D_FL*sys.jacobian(SlipFL.transpose(),
            dq);
1774
1775
1776        lst new_atom_list_dSBR_dq, new_exp_list_dSBR_dq;
1777        matrix_list_optimize (dSBR_dq, new_atom_list_dSBR_dq,
            new_exp_list_dSBR_dq);
1778        sys.export_Matrix_C ("dSBR_dq","_dSBR_dq",dSBR_dq,
            new_atom_list_dSBR_dq, new_exp_list_dSBR_dq ,ORDER);
1779        sys.export_function_MATLAB("dSBR_dq", "dSBR_dq_",dSBR_dq,
            new_atom_list_dSBR_dq, new_exp_list_dSBR_dq);
1780
1781        lst new_atom_list_dSBL_dq, new_exp_list_dSBL_dq;
1782        matrix_list_optimize (dSBL_dq, new_atom_list_dSBL_dq,
            new_exp_list_dSBL_dq);
1783        sys.export_Matrix_C ("dSBL_dq","_dSBL_dq",dSBL_dq,
            new_atom_list_dSBL_dq, new_exp_list_dSBL_dq ,ORDER);
1784        sys.export_function_MATLAB("dSBL_dq", "dSBL_dq_",dSBL_dq,
            new_atom_list_dSBL_dq, new_exp_list_dSBL_dq);
1785
1786        lst new_atom_list_dSFR_dq, new_exp_list_dSFR_dq;
1787        matrix_list_optimize (dSFR_dq, new_atom_list_dSFR_dq,
            new_exp_list_dSFR_dq);
1788        sys.export_Matrix_C ("dSFR_dq","_dSFR_dq",dSFR_dq,
            new_atom_list_dSFR_dq, new_exp_list_dSFR_dq ,ORDER);
1789        sys.export_function_MATLAB("dSFR_dq", "dSFR_dq_",dSFR_dq,
            new_atom_list_dSFR_dq, new_exp_list_dSFR_dq);
1790
1791        lst new_atom_list_dSFL_dq, new_exp_list_dSFL_dq;
1792        matrix_list_optimize (dSFL_dq, new_atom_list_dSFL_dq,
            new_exp_list_dSFL_dq);
1793        sys.export_Matrix_C ("dSFL_dq","_dSFL_dq",dSFL_dq,
            new_atom_list_dSFL_dq, new_exp_list_dSFL_dq ,ORDER);
1794        sys.export_function_MATLAB("dSFL_dq", "dSFL_dq_",dSFL_dq,
            new_atom_list_dSFL_dq, new_exp_list_dSFL_dq);
1795
1796    cout <<" CKalker Exportng"<<endl;
1797
1798        lst new_atom_list_CKBR, new_exp_list_CKBR;
1799        matrix_list_optimize (CKBR, new_atom_list_CKBR,
            new_exp_list_CKBR);
1800        sys.export_Matrix_C ("CKBR","_CKBR",CKBR, new_atom_list_CKBR,
            new_exp_list_CKBR ,ORDER);
1801        sys.export_function_MATLAB("CKBR", "CKBR_",CKBR,
            new_atom_list_CKBR, new_exp_list_CKBR);
1802
1803        lst new_atom_list_CKBL, new_exp_list_CKBL;
1804        matrix_list_optimize (CKBL, new_atom_list_CKBL,
            new_exp_list_CKBL);
1805        sys.export_Matrix_C ("CKBL","_CKBL",CKBL, new_atom_list_CKBL,
            new_exp_list_CKBL ,ORDER);
```

```
1806    sys.export_function_MATLAB("CKBL", "CKBL_",CKBL,
        new_atom_list_CKBL, new_exp_list_CKBL);
1807
1808    lst new_atom_list_CKFR, new_exp_list_CKFR;
1809    matrix_list_optimize (CKFR, new_atom_list_CKFR,
        new_exp_list_CKFR);
1810    sys.export_Matrix_C ("CKFR","_CKFR",CKFR, new_atom_list_CKFR,
        new_exp_list_CKFR ,ORDER);
1811    sys.export_function_MATLAB("CKFR", "CKFR_",CKFR,
        new_atom_list_CKFR, new_exp_list_CKFR);
1812
1813    lst new_atom_list_CKFL, new_exp_list_CKFL;
1814    matrix_list_optimize (CKFL, new_atom_list_CKFL,
        new_exp_list_CKFL);
1815    sys.export_Matrix_C ("CKFL","_CKFL",CKFL, new_atom_list_CKFL,
        new_exp_list_CKFL ,ORDER);
1816    sys.export_function_MATLAB("CKFL", "CKFL_",CKFL,
        new_atom_list_CKFL, new_exp_list_CKFL);
1817
1818
1819
1820
1821    lst new_atom_list_D_BR, new_exp_list_D_BR;
1822    matrix_list_optimize (D_BR, new_atom_list_D_BR,
        new_exp_list_D_BR);
1823    sys.export_Matrix_C ("D_BR","_D_BR",D_BR, new_atom_list_D_BR,
        new_exp_list_D_BR ,ORDER);
1824    sys.export_function_MATLAB("D_BR", "D_BR_",D_BR,
        new_atom_list_D_BR, new_exp_list_D_BR);
1825
1826
1827    lst new_atom_list_D_BL, new_exp_list_D_BL;
1828    matrix_list_optimize (D_BL, new_atom_list_D_BL,
        new_exp_list_D_BL);
1829    sys.export_Matrix_C ("D_BL","_D_BL",D_BL, new_atom_list_D_BL,
        new_exp_list_D_BL ,ORDER);
1830    sys.export_function_MATLAB("D_BL", "D_BL_",D_BL,
        new_atom_list_D_BL, new_exp_list_D_BL);
1831
1832
1833
1834    lst new_atom_list_D_FR, new_exp_list_D_FR;
1835    matrix_list_optimize (D_FR, new_atom_list_D_FR,
        new_exp_list_D_FR);
1836    sys.export_Matrix_C ("D_FR","_D_FR",D_FR, new_atom_list_D_FR,
        new_exp_list_D_FR ,ORDER);
1837    sys.export_function_MATLAB("D_FR", "D_FR_",D_FR,
        new_atom_list_D_FR, new_exp_list_D_FR);
1838
1839
1840    lst new_atom_list_D_FL, new_exp_list_D_FL;
1841    matrix_list_optimize (D_FL, new_atom_list_D_FL,
        new_exp_list_D_FL);
1842    sys.export_Matrix_C ("D_FL","_D_FL",D_FL, new_atom_list_D_FL,
        new_exp_list_D_FL ,ORDER);
1843    sys.export_function_MATLAB("D_FL", "D_FL_",D_FL,
        new_atom_list_D_FL, new_exp_list_D_FL);
```

```
1844
1845 // sustituir las fuerzas de kalker (inputs) por cero en Q y
     exportar el nuevo Q (Q2)
1846
1847     vector < symbol_numeric * > fk;
1848     fk.push_back( &Fx_BR );fk.push_back( &Fy_BR );fk.push_back( &
     Mz_BR );
1849     fk.push_back( &Fx_BL );fk.push_back( &Fy_BL );fk.push_back( &
     Mz_BL );
1850     fk.push_back( &Fx_FR );fk.push_back( &Fy_FR );fk.push_back( &
     Mz_FR );
1851     fk.push_back( &Fx_FL );fk.push_back( &Fy_FL );fk.push_back( &
     Mz_FL );
1852
1853     for (int i=0; (i < Qo.rows()); ++i) {
1854         Qo(i,0) = recursive_substitution (Qo(i,0),fk , 0);
1855     }
1856
1857
1858     lst new_atom_list_Q2, new_exp_list_Q2;
1859     matrix_list_optimize (Qo, new_atom_list_Q2,new_exp_list_Q2);
1860     sys.export_Matrix_C ("Q2","Q2_",Qo, new_atom_list_Q2,
     new_exp_list_Q2 ,ORDER);
1861     sys.export_function_MATLAB("Q2", "Q2_",Qo, new_atom_list_Q2,
     new_exp_list_Q2);
1862
1863
1864 //
     ********************************************************************************
1865 //  Export C code for Direct Simulation
1866 //
     ********************************************************************************
1867
1868
1869   cout << "Export C code for Direct Simulation " << endl;
1870   sys.export_Dynamic_Simulation(sys, ORDER , MAPLE);
1871
1872 //
     ********************************************************************************
1873 //  Export Point, Base and Frame Diagrams
1874 //
     ********************************************************************************
1875 #ifdef GRAPHVIZ
1876   cout << "Export Point, Base and Frame Diagrams" << endl;
1877
1878   sys.export_Graphviz_dot (  );
1879
1880   //Generate eps figure
1881       system("dot -Tps   base_diagram.dot -o plain_base_diagram.eps
     ");
1882       system("latex base_diagram.tex");
1883       system("dvips base_diagram.dvi -o base_diagram.eps");
1884   //Generate figure eps
```

```
1885        system("dot -Tps   point_diagram.dot -o plain_point_diagram.
        eps");
1886        system("latex point_diagram.tex");
1887        system("dvips point_diagram.dvi -o point_diagram.eps");
1888    //Generate eps figure
1889        system("dot -Tps   frame_diagram.dot -o plain_frame_diagram.
        eps");
1890        system("latex frame_diagram.tex");
1891        system("dvips base_diagram.dvi -o base_diagram.eps");
1892  #endif
1893
1894
1895
1896  //
        ********************************************************************************
1897  //  Export MATLAB environment vector
1898  //
        ********************************************************************************
1899
1900    cout << "Export ENVIRONMENT file " << endl;
1901
1902    sys.export_environment_m (  );
1903
1904
1905  //
        ********************************************************************************
1906  //  Export config.ini file
1907  //
        ********************************************************************************
1908
1909    cout << "Export config.ini file " << endl;
1910
1911    sys.export_config_ini (  );
1912
1913  //
        ********************************************************************************
1914  //  Export C++ code for Openscenegraph
1915  //
        ********************************************************************************
1916
1917    cout << "Export C++ code for Openscenegraph." << endl;
1918
1919    sys.export_open_scene_graph ( );
1920
1921  //
        ********************************************************************************
1922  //  Export State File
1923  //
        ********************************************************************************
```

```
1924
1925    cout << "Export State File" << endl;
1926
1927        lst state;
1928
1929        //state = x , theta, *dx, *dtheta, *ddx, *ddtheta;
1930
1931    sys.export_write_data_file_C(state);
1932
1933
1934  //
        ********************************************************************************
1935  //  Export grphics.gnuplot
1936  //
        ********************************************************************************
1937
1938    cout << "Export GNUPLOT file" << endl;
1939          sys.export_gnuplot ( state );
1940
1941  //
        ********************************************************************************
1942  //  Summary
1943  //
        ********************************************************************************
1944
1945                                        cout << "" << endl;
1946                                        cout << "
        ******************************************" << endl;
1947                                        cout << "** COMPILED WITH
        OPTIONS                          **" << endl;
1948                                        cout << "
        ******************************************" << endl;
1949  if ( METHOD == LAGRANGE)              { cout << "** Dynamic equations
        ===> LAGRANGE              **" << endl;  }
1950  else  if ( METHOD == VIRTUAL_POWER) { cout << "** Dynamic equations
        ===> VIRTUAL_POWER         **" << endl;  }
1951  if ( ORDER == CMO)                    { cout << "** Matrix Order
        ===> Col_MO                **" << endl;  }
1952  else  if ( ORDER == RMO)              { cout << "** Matrix Order
        ===> Row_MO                **" << endl;  }
1953  if ( MAPLE == MAPLE_OFF)              { cout << "** Maple
        ===> OFF                   **" << endl;  }
1954  else  if ( MAPLE == MAPLE_ON)         { cout << "** Maple
        ===> ON                    **" << endl;  }
1955                                        cout << "
        ******************************************" << endl;
1956
1957
1958
1959  //
        ********************************************************************************
1960  //  END program
```

```
1961 //
     ********************************************************************************
1962
1963 return 0;
1964
1965 }
```

Fichero de integración en MATLAB

```
1  clear all
2
3  tic;
4
5  global t
6  coord_init
7  coord_vect_init
8  vel_init
9  vel_vect_init
10 unknowns_init
11 unknowns_vect_init
12 param_init
13 param_vect_init
14 inputs_init
15 inputs_vect_init
16 %
17 global breaks_wheelFR
18 global coefs_wheelFR;
19
20 splines_data_wheel;
21 global breaks_wheelFR;
22 global coefs_wheelFR;
23 global breaks_wheelFL;
24 global coefs_wheelFL;
25 global breaks_wheelBR;
26 global coefs_wheelBR;
27 global breaks_wheelBL;
28 global coefs_wheelBL;
29
30 global breaks_railFR;
31 global coefs_railFR;
32 global breaks_railFL;
33 global coefs_railFL;
34 global breaks_railBR;
35 global coefs_railBR;
36 global breaks_railBL;
37 global coefs_railBL;
38
39
40 NEWTON_RAPHSON_TOLERANCE = 1.0e-14;
41 TRAPEZOIDAL_TOLERANCE = 1.0e-8;
42
43
44 INTEG = 'EULER'
```

```
 %INTEG  ='TRAPEZOIDAL'
METF='KALKER'
 %METF='VISCOSO'
 %METF='SINFUERZAS'
 t_final = 15;
 delta_t = 0.001;
 % alpha=0.001;


 Ct=1/delta_t

 %CT2= 0.083314;

 CT2=   0.0278*2;

 %CT2=   0;
 aFrame=0;
 Phi_=zeros(20,1);
 dPhi_dq_=zeros(20,36);
 MXdPhi_dqZero_=zeros(56,56);
 Qgamma_=zeros(56,1);
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 % Solution state in equilibrium
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 q=evalq;
 dq=zeros(length(q),1);
 lambda=evallambda;

 x0=[q;lambda]
 options= optimoptions('fsolve','TolFun',1e-14,'TolX',1e-14)
 x=fsolve(@F_equildq,x0,options)
 q=x(1:36,1);
 lambda=x(36+1:end,1);

 lambdaupdate

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 % Initial Position
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


  q(4)=q(4)-0.003;
 dq(1)=0*(-0.500409153694144);
 dq(2)=0*(-0.500409153694144);
 dq(9)=0;
 dq(15)=0;

 T_WhsF=CT2*dq(2);
 T_WhsB=CT2*dq(1);
 T_MOTF = 2.0;
 T_MOTB = 2.0;
 error_nr= sqrt(Phi(q,dq)'*Phi(q,dq));
 % aFrame=0.0698;

```

```
while error_nr > NEWTON_RAPHSON_TOLERANCE
     delta_q = - pinv(dPhi_dq(q,dq)) * Phi(q,dq);
     q = q + delta_q;
     error_nr= sqrt(Phi(q,dq)'*Phi(q,dq));
end


delta_dq = (pinv(dPhi_dq(q,dq)) * (Beta(q,dq) - (dPhi_dq(q,dq) * dq
    )));
dq = dq + delta_dq;

if strcmp(METF, 'KALKER')

 Kl_BL=q(15)*0.053125;
 Kl_BR=q(15)*0.053125;
 Kl_FL=q(15)*0.053125;
 Kl_FR=q(15)*0.053125
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

t=0;
n_q=size(q,1);
n_lambda=size(unknowns,1);

fid=fopen('sol.mat','w');
fprintf(fid,'%e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
    %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
    %e %e %e %e %e %e %e %e %e %e\n',t, bRailB,bRailF,xCh,yCh,zCh,
    cCh,aCh,bCh,bWhsB,cWhsB,aWhsB,xWhsB,yWhsB,zWhsB,bWhsF,cWhsF,
    aWhsF,xWhsF,yWhsF,zWhsF,thetaW_BL,uW_BL,uR_BL,thetaR_BL,
    thetaW_BR,uW_BR,uR_BR,thetaR_BR,thetaW_FL,uW_FL,uR_FL,thetaR_FL
    ,thetaW_FR,uW_FR,uR_FR,thetaR_FR,Fx_BR, Fy_BR, Mz_BR,Fx_BL,
    Fy_BL,Mz_BL,Fx_FL,Fy_FL,Mz_FL,Fx_FR,Fy_FR, Mz_FR);

fid1=fopen('kln.mat','w');
fprintf(fid1,'%e %e %e %e %e %e %e %e %e\n',t, Kl_BL,uR_BL,Kl_BR,
    uR_BR,Kl_FL,uR_FL,Kl_FR,uR_FR);

fid2=fopen('dsol.mat','w');
fprintf(fid2,'%e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
     %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e\n',t,
      dbRailB,dbRailF,dxCh,dyCh,dzCh,dcCh,daCh,dbCh,dbWhsB,dcWhsB,
    daWhsB,dxWhsB,dyWhsB,dzWhsB,dbWhsF,dcWhsF,daWhsF,dxWhsF,dyWhsF,
    dzWhsF,dthetaW_BL,duW_BL,duR_BL,dthetaR_BL,dthetaW_BR,duW_BR,
    duR_BR,dthetaR_BR,dthetaW_FL,duW_FL,duR_FL,dthetaR_FL,
    dthetaW_FR,duW_FR,duR_FR,dthetaR_FR);

 KalkerCoeffUpdate2;
 MXdPhi_dqZero_=MXdPhi_dqZero(q,dq);
 Qgamma_=Qgamma(q,dq);
 ddqlambda = MXdPhi_dqZero_ \ Qgamma_;
 ddq=ddqlambda(1:n_q,1);
 lambda=ddqlambda(n_q+1:n_lambda+n_q,1);

lambdaupdate
% Kalkersolution
%
 Kl_BL=q(15)*Ry_wBL;
```

```
 Kl_BR=q(15)*Ry_wBR;
 Kl_FL=q(15)*Ry_wFL;
 Kl_FR=q(15)*Ry_wFR;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Integration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while t < t_final
if t>1
 deltat=0.05;
end

dqant=dq;
T_WhsF=CT2*dq(2);
T_WhsB=CT2*dq(1);
KalkerCoeffUpdate2;
Solve_Dynamics2;

q=q+dqant*delta_t+0.5*(dq-dqant)*delta_t;

%Proj

error_nr= sqrt(Phi(q,dq)'*Phi(q,dq));


while error_nr > NEWTON_RAPHSON_TOLERANCE
    delta_q = - pinv(dPhi_dq(q,dq)) * Phi(q,dq);
    q = q + delta_q;
    error_nr= sqrt(Phi(q,dq)'*Phi(q,dq));
end


delta_dq = (pinv(dPhi_dq(q,dq)) * (Beta(q,dq) - (dPhi_dq(q,dq) * dq
    )));
dq = dq + delta_dq;


T_WhsF=CT2*dq(2);
T_WhsB=CT2*dq(1);

KalkerCoeffUpdate2;
% Kalkersolution
 Kl_BL=q(15)*Ry_wBL;
 Kl_BR=q(15)*Ry_wBR;
 Kl_FL=q(15)*Ry_wFL;
 Kl_FR=q(15)*Ry_wFR;

t=t+delta_t;


fprintf(fid,'%e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
    %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
    %e %e %e %e %e %e %e %e %e %e\n',t, bRailB,bRailF,xCh,yCh,zCh,
    cCh,aCh,bCh,bWhsB,cWhsB,aWhsB,xWhsB,yWhsB,zWhsB,bWhsF,cWhsF,
    aWhsF,xWhsF,yWhsF,zWhsF,thetaW_BL,uW_BL,uR_BL,thetaR_BL,
    thetaW_BR,uW_BR,uR_BR,thetaR_BR,thetaW_FL,uW_FL,uR_FL,thetaR_FL
```

```
    ,thetaW_FR,uW_FR,uR_FR,thetaR_FR,Fx_BR, Fy_BR, Mz_BR,Fx_BL,
    Fy_BL,Mz_BL,Fx_FL,Fy_FL,Mz_FL,Fx_FR,Fy_FR, Mz_FR);
fprintf(fid1,'%e %e %e %e %e %e %e %e %e\n',t, Kl_BL,uR_BL,Kl_BR,
    uR_BR,Kl_FL,uR_FL,Kl_FR,uR_FR);
fprintf(fid2,'%e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
     %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e\n',t,
      dbRailB,dbRailF,dxCh,dyCh,dzCh,dcCh,daCh,dbCh,dbWhsB,dcWhsB,
    daWhsB,dxWhsB,dyWhsB,dzWhsB,dbWhsF,dcWhsF,daWhsF,dxWhsF,dyWhsF,
    dzWhsF,dthetaW_BL,duW_BL,duR_BL,dthetaR_BL,dthetaW_BR,duW_BR,
    duR_BR,dthetaR_BR,dthetaW_FL,duW_FL,duR_FL,dthetaR_FL,
    dthetaW_FR,duW_FR,duR_FR,dthetaR_FR);

end

% end

fclose(fid2);
fclose(fid1);

fclose(fid);
tiempo1 = toc;

fprintf('El proceso ha tardado %d segundos', tiempo1);

end

if strcmp(METF, 'VISCOSO')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

t=0;
n_q=size(q,1);
n_lambda=size(unknowns,1);


fid=fopen('sol.mat','w');
 fprintf(fid,'%e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
     %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
     %e %e %e %e %e %e %e %e %e %e\n',t,  bRailB,bRailF,xCh,yCh,zCh
    ,cCh,aCh,bCh,bWhsB,cWhsB,aWhsB,xWhsB,yWhsB,zWhsB,bWhsF,cWhsF,
    aWhsF,xWhsF,yWhsF,zWhsF,thetaW_BL,uW_BL,uR_BL,thetaR_BL,
    thetaW_BR,uW_BR,uR_BR,thetaR_BR,thetaW_FL,uW_FL,uR_FL,thetaR_FL
    ,thetaW_FR,uW_FR,uR_FR,thetaR_FR,Fx_BR, Fy_BR, Mz_BR,Fx_BL,
    Fy_BL,Mz_BL,Fx_FL,Fy_FL,Mz_FL,Fx_FR,Fy_FR, Mz_FR);



Kalker_Kinematics_ = Kalker_Kinematics(q,dq);
%
    Fx_BR=Kalker_Kinematics_(1,1);
    Fy_BR=Kalker_Kinematics_(2,1);
    Mz_BR=Kalker_Kinematics_(3,1);
    Fx_BL=Kalker_Kinematics_(4,1);
    Fy_BL=Kalker_Kinematics_(5,1);
    Mz_BL=Kalker_Kinematics_(6,1);
    Fx_FL=Kalker_Kinematics_(7,1);
```

```
228         Fy_FL=Kalker_Kinematics_(8,1);
229         Mz_FL=Kalker_Kinematics_(9,1);
230         Fx_FR=Kalker_Kinematics_(10,1);
231         Fy_FR=Kalker_Kinematics_(11,1);
232         Mz_FR=Kalker_Kinematics_(12,1);
233
234
235     MXdPhi_dqZero_=MXdPhi_dqZero(q,dq);
236     Qgamma_=Qgamma(q,dq);
237     ddqlambda = MXdPhi_dqZero_ \ Qgamma_;
238     ddq=ddqlambda(1:n_q,1);
239
240     lambda=ddqlambda(n_q+1:n_lambda+n_q,1);
241
242     lambda1=lambda(1,:);
243     lambda2=lambda(2,:);
244     lambda3=lambda(3,:);
245     lambda4=lambda(4,:);
246     lambda5=lambda(5,:);
247     lambda6=lambda(6,:);
248     lambda7=lambda(7,:);
249     lambda8=lambda(8,:);
250     lambda9=lambda(9,:);
251     lambda10=lambda(10,:);
252     lambda11=lambda(11,:);
253     lambda12=lambda(12,:);
254     lambda13=lambda(13,:);
255     lambda14=lambda(14,:);
256     lambda15=lambda(15,:);
257     lambda16=lambda(16,:);
258     lambda17=lambda(17,:);
259     lambda18=lambda(18,:);
260     lambda19=lambda(19,:);
261     lambda20=lambda(20,:);
262
263
264
265     while t < t_final
266
267
268
269         [aW_BL,bW_BL,cW_BL,dW_BL,lW_BL]=spline_get_coefs(coefs_wheelBL,
            breaks_wheelBL, uW_BL);
270         [aR_BL,bR_BL,cR_BL,dR_BL,lR_BL]=spline_get_coefs(coefs_railBL,
            breaks_railBL, uR_BL);
271
272         [aW_BR,bW_BR,cW_BR,dW_BR,lW_BR]=spline_get_coefs(coefs_wheelBR,
            breaks_wheelBR, uW_BR);
273         [aR_BR,bR_BR,cR_BR,dR_BR,lR_BR]=spline_get_coefs(coefs_railBR,
            breaks_railBR, uR_BR);
274
275         [aW_FL,bW_FL,cW_FL,dW_FL,lW_FL]=spline_get_coefs(coefs_wheelFL,
            breaks_wheelFL, uW_FL);
276         [aR_FL,bR_FL,cR_FL,dR_FL,lR_FL]=spline_get_coefs(coefs_railFL,
            breaks_railFL, uW_FL);
277
```

```
278     [aW_FR,bW_FR,cW_FR,dW_FR,lW_FR]=spline_get_coefs(coefs_wheelFR,
        breaks_wheelFR, uW_FR);
279     [aR_FR,bR_FR,cR_FR,dR_FR,lR_FR]=spline_get_coefs(coefs_railFR,
        breaks_railFR, uR_FR);
280 %
281 %   if t<2
282 %       F_Ay=0;
283 % else
284 %
285 %     F_Ay=10000;
286 % if t<3
287 %           F_Ay=10000;
288 %       else
289 %
290 %     F_Ay=0;
291 %       end
292 % end
293 Kalker_Kinematics_ = Kalker_Kinematics(q,dq);
294
295     Fx_BR=Kalker_Kinematics_(1,1);
296     Fy_BR=Kalker_Kinematics_(2,1);
297     Mz_BR=Kalker_Kinematics_(3,1);
298     Fx_BL=Kalker_Kinematics_(4,1);
299     Fy_BL=Kalker_Kinematics_(5,1);
300     Mz_BL=Kalker_Kinematics_(6,1);
301     Fx_FL=Kalker_Kinematics_(7,1);
302     Fy_FL=Kalker_Kinematics_(8,1);
303     Mz_FL=Kalker_Kinematics_(9,1);
304     Fx_FR=Kalker_Kinematics_(10,1);
305     Fy_FR=Kalker_Kinematics_(11,1);
306     Mz_FR=Kalker_Kinematics_(12,1);
307 % %
308 MXdPhi_dqZero_=MXdPhi_dqZero(q,dq);
309 Qgamma_=Qgamma(q,dq);
310 ddqlambda = MXdPhi_dqZero_ \ Qgamma_;
311 ddq=ddqlambda(1:n_q,1);
312
313 lambda=ddqlambda(n_q+1:n_lambda+n_q,1);
314
315 lambda1=lambda(1,:);
316 lambda2=lambda(2,:);
317 lambda3=lambda(3,:);
318 lambda4=lambda(4,:);
319 lambda5=lambda(5,:);
320 lambda6=lambda(6,:);
321 lambda7=lambda(7,:);
322 lambda8=lambda(8,:);
323 lambda9=lambda(9,:);
324 lambda10=lambda(10,:);
325 lambda11=lambda(11,:);
326 lambda12=lambda(12,:);
327 lambda13=lambda(13,:);
328 lambda14=lambda(14,:);
329 lambda15=lambda(15,:);
330 lambda16=lambda(16,:);
331 lambda17=lambda(17,:);
332 lambda18=lambda(18,:);
```

```
lambda19=lambda(19,:);
lambda20=lambda(20,:);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if strcmp(INTEG, 'EULER')
    q  = q + (dq + 0.5 * ddq * delta_t ) * delta_t;
    dq = dq + ddq * delta_t;
% end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if strcmp(INTEG, 'TRAPEZOIDAL')
%      qn=q;
%   dqn=dq;
%      ddqn=ddq;
%
%      % Euler mejorado
%      q  = qn + (dqn + 0.5 * ddqn * delta_t ) * delta_t;
%      q = dqn + ddqn * delta_t;
%      error= sqrt((q-qn)'*(q-qn))+sqrt((dq-dqn)'*(dq-dqn));
%
%      iter=0;
%
%   while error > TRAPEZOIDAL_TOLERANCE
%
%   aux = MXdPhi_dqZero(q,dq) \ Qgamma(q,dq);
%         ddq = aux(1:36,1);
%         lambda = aux(37:69,1);
%
%         qn1=q;
%         dqn1=dq;
%         ddqn1=ddq;
%
%      q= qn +  0.5 * delta_t * ( dqn + dqn1 );
%      dq=dqn + 0.5 * delta_t * ( ddqn + ddqn1 );
%        error= sqrt((q-qn1)'*(q-qn1))+sqrt((dq-dqn1)'*(dq-dqn1));
%      iter=iter+1;
%   end
% end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



t = t +delta_t;


    [aW_BL,bW_BL,cW_BL,dW_BL,lW_BL]=spline_get_coefs(coefs_wheelBL,
    breaks_wheelBL, uW_BL);
    [aR_BL,bR_BL,cR_BL,dR_BL,lR_BL]=spline_get_coefs(coefs_railBL,
    breaks_railBL, uR_BL);

    [aW_BR,bW_BR,cW_BR,dW_BR,lW_BR]=spline_get_coefs(coefs_wheelBR,
    breaks_wheelBR, uW_BR);
```

```
383       [aR_BR,bR_BR,cR_BR,dR_BR,lR_BR]=spline_get_coefs(coefs_railBR,
          breaks_railBR, uR_BR);
384
385       [aW_FL,bW_FL,cW_FL,dW_FL,lW_FL]=spline_get_coefs(coefs_wheelFL,
          breaks_wheelFL, uW_FL);
386       [aR_FL,bR_FL,cR_FL,dR_FL,lR_FL]=spline_get_coefs(coefs_railFL,
          breaks_railFL, uW_FL);
387
388       [aW_FR,bW_FR,cW_FR,dW_FR,lW_FR]=spline_get_coefs(coefs_wheelFR,
          breaks_wheelFR, uW_FR);
389       [aR_FR,bR_FR,cR_FR,dR_FR,lR_FR]=spline_get_coefs(coefs_railFR,
          breaks_railFR, uR_FR);
390
391
392
393
394       dPhi_dq_=dPhi_dq(q,dq);
395       delta_q = - dPhi_dq_ \ Phi(q,dq);
396       q = q + delta_q;
397
398
399     %dPhi_dq_=dPhi_dq(q,dq);
400     delta_dq =dPhi_dq_ \ (Beta(q,dq) - (dPhi_dq_ * dq));
401     dq = dq + delta_dq;
402   Kalker_Kinematics_ = Kalker_Kinematics(q,dq);
403     Fx_BR=Kalker_Kinematics_(1,1);
404       Fy_BR=Kalker_Kinematics_(2,1);
405       Mz_BR=Kalker_Kinematics_(3,1);
406       Fx_BL=Kalker_Kinematics_(4,1);
407       Fy_BL=Kalker_Kinematics_(5,1);
408       Mz_BL=Kalker_Kinematics_(6,1);
409       Fx_FL=Kalker_Kinematics_(7,1);
410       Fy_FL=Kalker_Kinematics_(8,1);
411       Mz_FL=Kalker_Kinematics_(9,1);
412       Fx_FR=Kalker_Kinematics_(10,1);
413       Fy_FR=Kalker_Kinematics_(11,1);
414       Mz_FR=Kalker_Kinematics_(12,1);
415
416   fprintf(fid,'%e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
          %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
          %e %e %e %e %e %e %e %e %e %e\n',t, bRailB,bRailF,xCh,yCh,zCh,
          cCh,aCh,bCh,bWhsB,cWhsB,aWhsB,xWhsB,yWhsB,zWhsB,bWhsF,cWhsF,
          aWhsF,xWhsF,yWhsF,zWhsF,thetaW_BL,uW_BL,uR_BL,thetaR_BL,
          thetaW_BR,uW_BR,uR_BR,thetaR_BR,thetaW_FL,uW_FL,uR_FL,thetaR_FL
          ,thetaW_FR,uW_FR,uR_FR,thetaR_FR,Fx_BR, Fy_BR, Mz_BR,Fx_BL,
          Fy_BL,Mz_BL,Fx_FL,Fy_FL,Mz_FL,Fx_FR,Fy_FR, Mz_FR);
417
418
419
420   end
421
422
423   fclose(fid);
424
425
426   end
427
```

```
if  strcmp(METF, 'SINFUERZAS')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

t=0;
n_q=size(q,1);
n_lambda=size(unknowns,1);
      Fx_BR=0;
     Fy_BR=0;
     Mz_BR=0;
     Fx_BL=0;
     Fy_BL=0;
     Mz_BL=0;
     Fx_FL=0;
     Fy_FL=0;
     Mz_FL=0;
     Fx_FR=0;
     Fy_FR=0;
     Mz_FR=0;
fid=fopen('sol.mat','w');
 fprintf(fid,'%e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
      %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
      %e %e %e %e %e %e %e %e %e %e\n',t,  bRailB,bRailF,xCh,yCh,zCh
     ,cCh,aCh,bCh,bWhsB,cWhsB,aWhsB,xWhsB,yWhsB,zWhsB,bWhsF,cWhsF,
     aWhsF,xWhsF,yWhsF,zWhsF,thetaW_BL,uW_BL,uR_BL,thetaR_BL,
     thetaW_BR,uW_BR,uR_BR,thetaR_BR,thetaW_FL,uW_FL,uR_FL,thetaR_FL
     ,thetaW_FR,uW_FR,uR_FR,thetaR_FR,Fx_BR, Fy_BR, Mz_BR,Fx_BL,
     Fy_BL,Mz_BL,Fx_FL,Fy_FL,Mz_FL,Fx_FR,Fy_FR, Mz_FR);



Kalker_Kinematics_ = Kalker_Kinematics(q,dq);


MXdPhi_dqZero_=MXdPhi_dqZero(q,dq);
Qgamma_=Qgamma(q,dq);
ddqlambda = MXdPhi_dqZero_ \ Qgamma_;
ddq=ddqlambda(1:n_q,1);

lambda=ddqlambda(n_q+1:n_lambda+n_q,1);

lambda1=lambda(1,:);
lambda2=lambda(2,:);
lambda3=lambda(3,:);
lambda4=lambda(4,:);
lambda5=lambda(5,:);
lambda6=lambda(6,:);
lambda7=lambda(7,:);
lambda8=lambda(8,:);
lambda9=lambda(9,:);
lambda10=lambda(10,:);
lambda11=lambda(11,:);
lambda12=lambda(12,:);
lambda13=lambda(13,:);
lambda14=lambda(14,:);
lambda15=lambda(15,:);
```

```
lambda16=lambda(16,:);
lambda17=lambda(17,:);
lambda18=lambda(18,:);
lambda19=lambda(19,:);
lambda20=lambda(20,:);



while t < t_final


    [aW_BL,bW_BL,cW_BL,dW_BL,lW_BL]=spline_get_coefs(coefs_wheelBL,
    breaks_wheelBL, uW_BL);
    [aR_BL,bR_BL,cR_BL,dR_BL,lR_BL]=spline_get_coefs(coefs_railBL,
    breaks_railBL, uR_BL);

    [aW_BR,bW_BR,cW_BR,dW_BR,lW_BR]=spline_get_coefs(coefs_wheelBR,
    breaks_wheelBR, uW_BR);
    [aR_BR,bR_BR,cR_BR,dR_BR,lR_BR]=spline_get_coefs(coefs_railBR,
    breaks_railBR, uR_BR);

    [aW_FL,bW_FL,cW_FL,dW_FL,lW_FL]=spline_get_coefs(coefs_wheelFL,
    breaks_wheelFL, uW_FL);
    [aR_FL,bR_FL,cR_FL,dR_FL,lR_FL]=spline_get_coefs(coefs_railFL,
    breaks_railFL, uW_FL);

    [aW_FR,bW_FR,cW_FR,dW_FR,lW_FR]=spline_get_coefs(coefs_wheelFR,
    breaks_wheelFR, uW_FR);
    [aR_FR,bR_FR,cR_FR,dR_FR,lR_FR]=spline_get_coefs(coefs_railFR,
    breaks_railFR, uR_FR);
%
%   if t<2
%         F_Ay=0;
% else
%
%     F_Ay=10000;
% if t<3
%             F_Ay=10000;
%         else
%
%     F_Ay=0;
%         end
% end

MXdPhi_dqZero_=MXdPhi_dqZero(q,dq);
Qgamma_=Qgamma(q,dq);
ddqlambda = MXdPhi_dqZero_ \ Qgamma_;
ddq=ddqlambda(1:n_q,1);

lambda=ddqlambda(n_q+1:n_lambda+n_q,1);

lambda1=lambda(1,:);
lambda2=lambda(2,:);
lambda3=lambda(3,:);
lambda4=lambda(4,:);
lambda5=lambda(5,:);
```

```
lambda6=lambda(6,:);
lambda7=lambda(7,:);
lambda8=lambda(8,:);
lambda9=lambda(9,:);
lambda10=lambda(10,:);
lambda11=lambda(11,:);
lambda12=lambda(12,:);
lambda13=lambda(13,:);
lambda14=lambda(14,:);
lambda15=lambda(15,:);
lambda16=lambda(16,:);
lambda17=lambda(17,:);
lambda18=lambda(18,:);
lambda19=lambda(19,:);
lambda20=lambda(20,:);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if strcmp(INTEG, 'EULER')
    q  = q + (dq + 0.5 * ddq * delta_t ) * delta_t;
    dq = dq + ddq * delta_t;
% end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if strcmp(INTEG, 'TRAPEZOIDAL')
%      qn=q;
%   dqn=dq;
%      ddqn=ddq;
%
%      % Euler mejorado
%      q  = qn + (dqn + 0.5 * ddqn * delta_t ) * delta_t;
%      q = dqn + ddqn * delta_t;
%      error= sqrt((q-qn)'*(q-qn))+sqrt((dq-dqn)'*(dq-dqn));
%
%      iter=0;
%
%   while error > TRAPEZOIDAL_TOLERANCE
%
%   aux = MXdPhi_dqZero(q,dq) \ Qgamma(q,dq);
%          ddq = aux(1:36,1);
%          lambda = aux(37:69,1);
%
%          qn1=q;
%          dqn1=dq;
%          ddqn1=ddq;
%
%      q= qn +  0.5 * delta_t * ( dqn + dqn1 );
%      dq=dqn + 0.5 * delta_t * ( ddqn + ddqn1 );
%        error= sqrt((q-qn1)'*(q-qn1))+sqrt((dq-dqn1)'*(dq-dqn1));
%      iter=iter+1;
%   end
% end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
580
581
582 t = t +delta_t;
583
584
585     [aW_BL,bW_BL,cW_BL,dW_BL,lW_BL]=spline_get_coefs(coefs_wheelBL,
        breaks_wheelBL, uW_BL);
586     [aR_BL,bR_BL,cR_BL,dR_BL,lR_BL]=spline_get_coefs(coefs_railBL,
        breaks_railBL, uR_BL);
587
588     [aW_BR,bW_BR,cW_BR,dW_BR,lW_BR]=spline_get_coefs(coefs_wheelBR,
        breaks_wheelBR, uW_BR);
589     [aR_BR,bR_BR,cR_BR,dR_BR,lR_BR]=spline_get_coefs(coefs_railBR,
        breaks_railBR, uR_BR);
590
591     [aW_FL,bW_FL,cW_FL,dW_FL,lW_FL]=spline_get_coefs(coefs_wheelFL,
        breaks_wheelFL, uW_FL);
592     [aR_FL,bR_FL,cR_FL,dR_FL,lR_FL]=spline_get_coefs(coefs_railFL,
        breaks_railFL, uW_FL);
593
594     [aW_FR,bW_FR,cW_FR,dW_FR,lW_FR]=spline_get_coefs(coefs_wheelFR,
        breaks_wheelFR, uW_FR);
595     [aR_FR,bR_FR,cR_FR,dR_FR,lR_FR]=spline_get_coefs(coefs_railFR,
        breaks_railFR, uR_FR);
596
597
598
599
600     dPhi_dq_=dPhi_dq(q,dq);
601     delta_q = - dPhi_dq_ \ Phi(q,dq);
602     q = q + delta_q;
603
604
605   %dPhi_dq_=dPhi_dq(q,dq);
606   delta_dq =dPhi_dq_ \ (Beta(q,dq) - (dPhi_dq_ * dq));
607   dq = dq + delta_dq;
608
609
610
611     Fx_BR=0;
612      Fy_BR=0;
613      Mz_BR=0;
614     Fx_BL=0;
615      Fy_BL=0;
616      Mz_BL=0;
617     Fx_FL=0;
618      Fy_FL=0;
619      Mz_FL=0;
620     Fx_FR=0;
621      Fy_FR=0;
622      Mz_FR=0;
623
624
625
626 fprintf(fid,'%e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
    %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e
    %e %e %e %e %e %e %e %e %e %e\n',t, bRailB,bRailF,xCh,yCh,zCh,
```

```
        cCh, aCh, bCh, bWhsB, cWhsB, aWhsB, xWhsB, yWhsB, zWhsB, bWhsF, cWhsF,
        aWhsF, xWhsF, yWhsF, zWhsF, thetaW_BL, uW_BL, uR_BL, thetaR_BL,
        thetaW_BR, uW_BR, uR_BR, thetaR_BR, thetaW_FL, uW_FL, uR_FL, thetaR_FL
        , thetaW_FR, uW_FR, uR_FR, thetaR_FR, Fx_BR,  Fy_BR,  Mz_BR, Fx_BL,
        Fy_BL, Mz_BL, Fx_FL, Fy_FL, Mz_FL, Fx_FR, Fy_FR,  Mz_FR);
627
628
629
630 end
631
632
633 fclose(fid);
634
635
636 end
```

Fichero Solve Dynamics2

```
%Solve_Dynamics
dPhi_dqSize=size(dPhi_dq(q,dq));
q_size = length(q);
Zeros=zeros(dPhi_dqSize(1));


    [aW_BL,bW_BL,cW_BL,dW_BL,lW_BL]=spline_get_coefs(coefs_wheelBL,
    breaks_wheelBL, uW_BL);
    [aR_BL,bR_BL,cR_BL,dR_BL,lR_BL]=spline_get_coefs(coefs_railBL,
    breaks_railBL, uR_BL);

    [aW_BR,bW_BR,cW_BR,dW_BR,lW_BR]=spline_get_coefs(coefs_wheelBR,
    breaks_wheelBR, uW_BR);
    [aR_BR,bR_BR,cR_BR,dR_BR,lR_BR]=spline_get_coefs(coefs_railBR,
    breaks_railBR, uR_BR);

    [aW_FL,bW_FL,cW_FL,dW_FL,lW_FL]=spline_get_coefs(coefs_wheelFL,
    breaks_wheelFL, uW_FL);
    [aR_FL,bR_FL,cR_FL,dR_FL,lR_FL]=spline_get_coefs(coefs_railFL,
    breaks_railFL, uW_FL);

    [aW_FR,bW_FR,cW_FR,dW_FR,lW_FR]=spline_get_coefs(coefs_wheelFR,
    breaks_wheelFR, uW_FR);
    [aR_FR,bR_FR,cR_FR,dR_FR,lR_FR]=spline_get_coefs(coefs_railFR,
    breaks_railFR, uR_FR);

%MQn = MQ(q,dq,t,param,input);
Mn = M(q,dq);
%Qn = Q(q,dq,t,param,input);
Q2n= Q2(q,dq);
CKBRn= CKBR(q,dq);
CKBLn= CKBL(q,dq);
CKFRn= CKFR(q,dq);
CKFLn= CKFL(q,dq);
dPhi_dqn= dPhi_dq(q,dq);
Gamman = Gamma(q,dq);

% Output_=Output(q,dq,ddq,unkn,t,param,input);
 Radius_BR_ = Radius_BR(q,dq);
 Radius_BL_ = Radius_BL(q,dq);
 Radius_FR_ = Radius_FR(q,dq);
 Radius_FL_ = Radius_FL(q,dq);

 tolr=1e-5;


if Radius_BR_(9,1)<tolr
velBR=tolr;
velBL=tolr;
velFR=tolr;
velFL=tolr;
else
velBR=Radius_BR_(9,1);
velBL=Radius_BL_(9,1);
velFR=Radius_FR_(9,1);
```

```
velFL=Radius_FL_(9,1);

end

%
%MX=[Mn, dPhi_dqn'; dPhi_dqn, Zeros];
%QG=[Q2n-(CKBRn/velBR + CKBLn/velBL+ CKFRn/velFR + CKFLn/velFL)*dq
    ;Gamman];
CKT=(CKBRn/velBR + CKBLn/velBL+ CKFRn/velFR + CKFLn/velFL);
MX=[Mn+(CKT*delta_t),(dPhi_dq(q,dq)'*delta_t);dPhi_dqn, Zeros];

QG=[(Q2n*delta_t)+(Mn*dq);(Gamman*delta_t)+(dPhi_dqn*dq)];


dqlambda=pinv(MX)*QG;
dq=dqlambda(1:q_size);
unkn=dqlambda(q_size+1:length(dqlambda));


Forces_Vcreep_BL1=D_BL(q,dq)*dSBL_dq(q,dq)*dq/velBL;
Forces_Vcreep_BR1=D_BR(q,dq)*dSBR_dq(q,dq)*dq/velBR;
Forces_Vcreep_FL1=D_FL(q,dq)*dSFL_dq(q,dq)*dq/velFL;
Forces_Vcreep_FR1=D_FR(q,dq)*dSFR_dq(q,dq)*dq/velFR;


    Fx_BR=Forces_Vcreep_BR1(1,1);
    Fy_BR=Forces_Vcreep_BR1(2,1);
    Mz_BR=Forces_Vcreep_BR1(3,1);
    Fx_BL=Forces_Vcreep_BL1(1,1);
    Fy_BL=Forces_Vcreep_BL1(2,1);
    Mz_BL=Forces_Vcreep_BL1(3,1);
    Fx_FL=Forces_Vcreep_FL1(1,1);
    Fy_FL=Forces_Vcreep_FL1(2,1);
    Mz_FL=Forces_Vcreep_FL1(3,1);
    Fx_FR=Forces_Vcreep_FR1(1,1);
    Fy_FR=Forces_Vcreep_FR1(2,1);
    Mz_FR=Forces_Vcreep_FR1(3,1);
```

Fichero KalkerCoeffUpdate2

```
% KalkerCoeffUpdate

 Radius_BR_ = Radius_BR(q,dq);
 Radius_BL_ = Radius_BL(q,dq);
 Radius_FR_ = Radius_FR(q,dq);
 Radius_FL_ = Radius_FL(q,dq);


  Ry_wBR = Radius_BR_(1,1);
  Rx_wBR = Radius_BR_(2,1);
  Ry_rBR = Radius_BR_(3,1);
  Rx_rBR = Radius_BR_(4,1);

   N_BR=Radius_BR_(5,1);

  R_BR = [Rx_wBR,Rx_rBR,Ry_wBR,Ry_rBR];

  Ry_wBL = Radius_BL_(1,1);
  Rx_wBL = Radius_BL_(2,1);
  Ry_rBL = Radius_BL_(3,1);
  Rx_rBL = Radius_BL_(4,1);
  N_BL=Radius_BL_(5,1);
   R_BL = [Rx_wBL,Rx_rBL,Ry_wBL,Ry_rBL];

  N_FL=Radius_FL_(5,1);

  Ry_wFL = Radius_FL_(1,1);
  Rx_wFL = Radius_FL_(2,1);
  Ry_rFL = Radius_FL_(3,1);
  Rx_rFL = Radius_FL_(4,1);
  R_FL = [Rx_wFL,Rx_rFL,Ry_wFL,Ry_rFL];
% Output_=Output(q,dq,ddq,unkn,t,param,input);

N_FR=Radius_FR_(5,1);

  Ry_wFR = Radius_FR_(1,1);
  Rx_wFR = Radius_FR_(2,1);
  Ry_rFR = Radius_FR_(3,1);
  Rx_rFR = Radius_FR_(4,1);
 R_FR = [Rx_wFR,Rx_rFR,Ry_wFR,Ry_rFR];

[aBR,bBR] = Hertz_ellipse( N_BR , R_BR ,param( 59 ),param( 60 ));
param( 62:63 ) = [aBR,bBR];
param( 70:73 ) = kalker_coeff(aBR,bBR,param( 60 ));

[aBL,bBL] = Hertz_ellipse( N_BL , R_BL ,param( 59 ),param( 60 ));
param( 64:65 ) = [aBL,bBL];
param( 74:77 ) = kalker_coeff(aBL,bBL,param( 60 ));

[aFR,bFR] = Hertz_ellipse( N_FR , R_FR ,param( 59 ),param( 60 ));
param( 66:67 ) = [aFR,bFR];
param( 78:81 ) = kalker_coeff(aFR,bFR,param( 60 ));

[aFL,bFL] = Hertz_ellipse( N_FL , R_FL ,param( 59 ),param( 60 ));
param( 68:69 ) = [aFL,bFL];
```

```
56 param( 82:85 ) = kalker_coeff(aFL,bFL,param( 60 ));
57
58  C11BR=param(70);
59  C22BR=param(71);
60  C23BR=param(72);
61  C33BR=param(73);
62  C11BL=param(74);
63  C22BL=param(75);
64  C23BL=param(76);
65  C33BL=param(77);
66  C11FR=param(78);
67  C22FR=param(79);
68  C23FR=param(80);
69  C33FR=param(81);
70  C11FL=param(82);
71  C22FL=param(83);
72  C23FL=param(84);
73  C33FL=param(85);
74
75     if   bRailB <0
76
77  C23BR=-C23BR;
78  C23BL=-C23BL;
79  C23FR=-C23FR;
80  C23FL=-C23FL;
81
82     end
```

Fichero de Cálculo de Splines en MATLAB

```
1
2  clear all
3  close all
4
5
6  x0
       =[-11.6363438450441;-11.4395488186953;-11.0602733846826;-7.97808358844242;-4.03387719528
7  options=optimoptions('fminunc','Algorithm','quasi-newton', 'MaxIter
       ',2000,'TolFun',1e-8,'TolX',1e-8,'MaxFunEvals',2000000)
8  x=fminunc(@object,x0,options);
9  %x=fminunc(@object,x0)
10
11 for i=1:length(x);
12     yy(i)=myfun(x(i));
13 end
14
15 z=yy/1000;
16 r=x/1000
17  sy=spline(r,z);
18
19
20
21
22  hold on
23
24 xx=[-11.67:0.001:11.67];
25
26 for i=1:length(xx);
27     yy1(i)=myfun(xx(i));
28 end
29 z1=yy1/1000;
30 r1=xx/1000
31
32
33 yyspline=ppval(sy,r1);
34 plot(r1,z1,'r')
35 axis equal
36 plot(r1,yyspline,'b')
37 plot(r,z,'o')
38 error1=norm(yyspline-yy1);
```